1972

A PL360-based compiler generating system.

Woods, Robert Allen.

Monterey, California. Naval Postgraduate School

A PL360-BASED COMPILER GENERATING SYSTEM

Robert Allen Woods

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A PL360-BASED COMPILER GENERATING SYSTEM

by

Robert Allen Woods

Thesis Advisor: R. H. Brubaker

December 1972

Approved for public release; distribution unlimited.

A PL360-Based Compiler Generating System

by

Robert Allen Woods
Lieutenant, United States Navy
B.S., Kansas State University, 1965

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1972

ABSTRACT

A compiler generating system written in the language PL360 to run on IBM System/360 computers is presented. The concepts and principles of the XPL compiler generating system are reviewed. The SLR(k) parsing algorithm is briefly described, and an example of SLR(l) parsing is presented. A description of the compiler generating system is presented along with its limitations, and instructions for its use are given. The required PL360 to System/360 interface is described and a listing is included. Program listings and sample input and output are included in the appendices.

TABLE OF CONTENTS

I. INTRODUCTION

Most computer programs are not written in a machine language. Therefore, it is necessary that a program (assembler, compiler) first translate the input sentences of the source program into an appropriate sequence of machine instructions before execution can begin. The task of writing this program, or translator, is often a long and tedious process. Computer Science research and experience with existing translators have made it possible for researchers to automate major portions of the task.

Most translators have many tasks in common, such as scanning text, analyzing syntax, synthesizing code, and interacting with an operating system. It is these tasks that are automated by translator writing systems (TWSs). Once these tasks are handled, the compiler writer can concentrate on those items unique to his particular translator.

The objective of the research reported herein was to complete the development of a TWS based upon the language PL360 [Ref. 1] and to implement the completed TWS at the W. R. Church Computer Center, Naval Postgraduate School. This goal has been achieved by completion of the protocompiler (a syntax checker without code synthesis facilities, a prototype compiler) originally written by Blanchard [Ref. 2] and the development of a PL360 program to analyze a grammar and produce the corresponding tables required by the SLR(1) [Ref. 3] parsing algorithm.

The next chapter of this report contains a review of one TWS and a description of the SLR(k) parsing algorithm. A brief description of the language PL360 is also presented to provide the reader with some background information. Chapter III contains a description of the completed PL360 compiler generating system. The required OS/360 interface is presented in Chapter IV.

II. BACKGROUND

This chapter first explores the concepts and principles of one TWS and then presents a review of the parsing algorithm used for the PL360-based TWS. The translator writing system reviewed is the XPL system of McKeeman, et al. [Ref. 4]. The parsing algorithm, SLR(k), is that chosen by Blanchard as the basis for the proto-compiler. An excellent paper by Feldman and Gries [Ref. 5] contains a critical review of many TWS efforts. Section C contains background information concerning the language PL360.

A. THE XPL SYSTEM

In this section, the principles of McKeeman's compiler generator are discussed. The system is explained in detail in Ref. 4, which serves both as an introduction to the construction of TWSs and as a user's manual for the XPL programming language.

The parsing algorithm used by McKeeman is the mixed-strategy (MSP) algorithm, a particular type of bottom-up parser, which is a modification of Wirth's [Ref. 6] precedence concept. The distinguishing feature of the algorithm is that it does not use state-of-the-parse information, as top-down methods do; rather, it involves examining the canonical sentential form (each string in a canonical parse) to determine what unique parse step is applicable and then performs a substitution.

The three major programs of the XPL compiler generating system are the syntax analyzer which builds the tables required by the MSP algorithm, the proto-compiler with which the user can produce a compiler, and the XPL compiler which translates XPL statements to System/360 machine code.

The syntax analyzer is a program which accepts the BNF definition of a grammar, determines whether the grammer is MSP(2,1:1,1) (a special case of MSP), constructs parsing decision tables, and punches those tables on cards in the form of XPL declarations.

The proto-compiler uses the cards produced by the analyzer and functions as a syntax checker. The user may build on the proto-compiler by rewriting the code synthesis routine to implement the semantics of the new language to be compiled and by altering the text-scanning routine to process the terminal symbols of the new language. During syntax analysis each reduction causes the code synthesis procedure to be invoked, and the appropriate machine code can be generated.

The XPL compiler generating system allows the user to construct compilers for languages described by grammars with a minimum of effort.

B. THE SLR(k) PARSING ALGORITHM

DeRemer defines a class of context-free grammars called "Simple LR(k)" or SLR(k) which includes the simple precedence grammars as proper subsets. A method for constructing parsers for SLR(k) grammars is given in Ref. 9. SLR(k) parsers have been implemented by DeRemer and appear to be

superior to corresponding MSP parsers both in the speed of parser construction and in the size and speed of the resulting parsers.

For the stacking decision, the MSP algorithm uses at most the top two symbols on the parse stack and the next symbol from the input text. SLR(k) parsers make the stacking decision based on all the symbols in the parse stack plus k more from the input text. This is accomplished by restructuring the stack and saving state-of-the-parse information. The operation of the parser will be illustrated by example.

Consider a sample grammar:

G : : = E
E : : = E + T | T
T : : = (E) | x

The finite state machine represented in Fig. 1 parses sentences generated by the sample grammar. The algorithm is started in state 0 and passes through a series of states until reaching a state with no successor. The indicated rule is applied and the parser is restarted in state 0. The algorithm terminates upon reaching state 2 and encountering an end-of-file mark. For example, when in state 1 and the symbol "x" is encountered, apply the reduction T x and restart; when in state 3 and the symbol "(" is encountered, stack the symbol and enter state 4.

The SLR(1) parsing algorithm has been implemented in Blanchards proto-compiler.

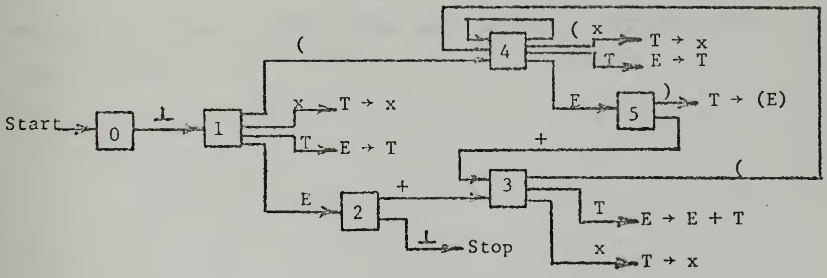

Fig. 1 Finite State Machine for the Sample Grammar

C. PL360

In 1968, Wirth published a formal description of PL360 [Ref. 1], a language designed specifically for the IBM System/360. A year later, a compiler for PL360 was written by Wirth, J. W. Wells, Jr., and E. Satterthwaite, Jr. and made available through the IBM Contributed Program Library [Ref. 7]. Several amendments to the original language definition were included with the documentation issued with the compiler. Further extensions and modifications to the language have recently been carried out, most notably by M. A. Malcolm [Ref. 8]. Malcolm's PL360 manual has incorporated all changes to the language definition and compiler description to date.

PL360 is a language that provides most of the facilities provided by System/360 Assembler Language yet exhibits a block-structured, ALGOL-like syntax. It was designed to

improve the readability of programs which take advantage of features unique to the System/360. The PL360 language is defined by a set of BNF rules and semantic explanations given in Ref. 8.

III. DESCRIPTION OF THE PL360 COMPILER GENERATING SYSTEM

This chapter describes the completed compiler generating system that was begun by Blanchard. To complete the system it was necessary to:

1. Implement in PL360 a syntax analyzer that would provide the necessary SLR(1) parsing tables required for the proto-compiler.
2. Make several changes to the proto-compiler to make it more versatile.

A. THE SYNTAX ANALYZER

1. Description of the Syntax Analyzer

The program called SLR1ANAL is the grammar analyzer of the compiler generating system. A program listing is contained in Appendix A. It is patterned after DeRemer's Simple LR(1) grammar analyzer that was written in XPL and has the same function and basic structure.

SLR1ANAL constructs a Characteristic Finite State Machine for a grammar and produces the SLR(1) parsing tables that are required by the proto-compiler. Appendix B contains a listing of a sample input grammar. The output of the SLR(1) parsing tables is in the form of punched PL360 declarations. Appendix C contains a complete listing of the SLR1ANAL output for the input shown in Appendix B.

The program can be divided into 7 basic parts:

1. The data area which contains 9 PL360 data segments that contain the numerous arrays that are required.
2. The procedure READG that reads the input BNF description of a grammar, sorts the grammar, finds the goal symbol of the grammar and prints the grammar in standard BNF format.
3. The procedure COMPUTECFSM that computes and outputs the Characteristic Finite State Machine.
4. The procedure LOOKAHEAD that computes and outputs, if necessary, the required look-ahead sets for the SLR(1) parser.
5. The procedure COMPUTEDPDA that computes and outputs the look-back states and sets the successors of the reduce states.
6. The procedure PUNCHDPDA that prints and punches the declarations required for the proto-compiler.
7. The main program that controls the logical flow of the computation.

2. How to Use the Syntax Analyzer

Input to SLR1ANAL is made by cards containing BNF productions which are placed one to a card by using the following conventions: if the first column is non-blank, the first token is taken to be the left part of the production; otherwise, the left part is assumed to be the same as the left part of the preceding production. The balance of the

card is taken to be the right part of the production. Any token that does not occur as a left part is a terminal symbol. Any token that occurs only as a left part is taken to be the goal symbol for the grammar. All productions with the same left part must be grouped together. If the user wishes to recognize identifiers and/or numbers he should include as terminal symbols <IDENTIFIER> and <NUMBER>, respectively. The proto-compiler has mechanisms to recognize these symbols and take the necessary actions.

Input cards with the character "@" in the first column are treated as comment or control cards as listed below. It is possible to batch grammars together by separating the grammars by control cards beginning with "@EOG". Such a card should not be included after the last grammar. The SLR1ANAL control cards are as follows:

a. @I is a toggle controlling the listing of the input data cards (initially off).

b. @G is a toggle controlling the listing of the reformatted grammar (initially on).

c. @C is a toggle controlling the listing of the configuration sets (initially off).

d. @F is a toggle controlling the listing of the CFSM (initially on).

e. @L is a toggle controlling the listing of the look-ahead sets (initially on).

f. @D is a toggle controlling the listing of the DPDA (initially on).

g. @P is a toggle controlling the punching of the parsing tables (initially off).

h. @EOG separates batched grammars.

i. All other cards containing "@" in column one are taken as comment cards.

The job control language required to execute SLR1ANAL can be found in Appendix G.

3. Limitations of the Syntax Analyzer

SLR1ANAL reads a grammar and attempts to construct an SLR(1) parser for it. The program either succeeds and punches tables that represent the parser, or it prints messages stating the reasons why it cannot. If the program fails, the given grammar is either too large in some aspect or it is not SLR(1); i. e., the program will suceed, except for space limitation, for any SLR(1) grammar. The one limitation most notable is that the BNF description of an input grammar cannot exceed 250 productions.

B. THE PROTO-COMPILER

1. Description of the Proto-Compiler

The program called "PROTOCOM" forms the basis of the PL360 compiler generator system. A listing of PROTOCOM is given in Appendix D. It is patterned after McKeeman's proto-compiler and has the same function and basic structure.

PROTOCOM uses the SLR(1) parsing tables obtained from SLR1ANAL. The tables are referenced by the algorithm contained in the procedure ANALYZE to implement the finite state machine of a grammar.

ANALYZE calls on procedure PUSHANDREAD or procedure SYNTHESIZE based on a decision to stack the current symbol and read a new one or to make a reduction and perform the required semantic operations. Since code synthesis is not a function of PROTOCOM, SYNTHESIZE exists only to maintain flow of control and indicate that its presence would be required in a full-scale compiler.

Procedure SCAN, called from either PUSHANDREAD or ANALYZE, interprets characters in the card buffer. Upon reaching the end of a card image, a call on procedure GETCARD causes a new card to be read.

The only other major procedure is ERROR which is called from a number of other routines. It handles syntax errors and prints diagnostic messages.

PROTOCOM as implemented by Blanchard would accept as terminal symbols only those consisting of one character, with the possibility of future expansion to eight characters. One other limitation was the data area available for the parser tables. The mechanisms causing these limitations were changed to allow the proto-compiler to be useful for large grammars. PROTOCOM will now recognize all symbols of 64 characters or less, and the data area available for the parsing tables is limited only by the space available in the user's System/360 region.

2. How to Use the Proto-Compiler

The first step in using PROTOCOM is to place the punched declarations from SLR1ANAL into the beginning

of PROTOCOM. They must be placed after the comment "THE FOLLOWING CARDS WERE PUNCHED BY THE SLR(1) SYNTAX ANALYZER" and before the comment "END OF CARDS PUNCHED BY SLR(1) SYNTAX ANALYZER." The cards that are already present between the two comments must be removed.

Input to PROTOCOM is a program written in the language defined by the grammar. PROTOCOM will parse the input program. If it detects any conflicts between the input program and the SLR(1) parsing tables, it will output an appropriate error message along with a partial parse. If no errors are found, the only output is a listing of the input program and the message "END OF COMPILATION." A listing of a sample output is located in Appendix E.

The job control language required to execute PROTOCOM is given in Appendix G.

3. Limitations of the Proto-Compiler

PROTOCOM appears to have no limitation as a syntax checker. If the user wishes to use PROTOCOM as the basis for a compiler, he must write the procedures SYNTHESIZE and EMIT. These procedures are necessary to generate the machine code necessary to execute the program. At present PROTOCOM does not build any symbol tables. Therefore, the user must also write the procedure LOOKUP to enter and look up symbols in the symbol table.

IV. OPERATING SYSTEM INTERFACE

Since PL360 object modules do not communicate directly with an operating system, an interface program must be provided that can be compiled separately and linked to the PL360 object module by the linkage editor or linking loader.

A program called "PLIO" is presently being used as the interface between the PL360 object modules and OS/360. A listing of PLIO can be found in Appendix F. The current interface contains 4 subroutines which facilitate input and output operations. In addition to these 4 subroutines, 2 more subroutines need to be added to PLIO if PROTOCOM is to be used as a compiler. The first subroutine would decode any required parameter list, open required data sets, obtain free storage, and supply system identification. The second would release free storage and close the required data sets.

The subroutines presently in PLIO are listed below with their names and specifications.

A. READ: Read an 80 character record from the input data set and store it in the 80-byte area designated by the address contained in R0. If an end-of-file is encountered, then set the condition code to 2, else set it to 0.

B. WRITE: Write a 132 bype record from the memory location designated by the address contained in RO to the output data set.

C. PAGE: Causes the next output record to contain the USASI control character "1" to be placed into the first position of the next output record.

D. PUNCH: Write the 80-byte record whose address is contained in R0 to the punch output data set.

PLIO uses the following input and output data sets, which are identified by their DDNAMES. All data sets are sequential with fixed block format.

A. SYSIN: This data set contains the input and is referenced by the READ subroutine. The data set consist of compiler instructions and one or more source programs.

B. SYSOUT: This data set contains output originating with the subroutine WRITE and any compiler diagnostic messages. The logical record length is 133 bytes.

C. SYSPUNCH: This data set contains output originating with the subroutine PUNCH. The logical record is 80 bytes.

If PROTOCOM is used as a compiler then one data set with a logical record length of 80 bytes and closed with a disposition of REREAD would be required to contain object module output. The data set SYSPUNCH could be used for this purpose and passed to the following step (i.e., LINK).

V. CONCLUSIONS

The PL360 language is well suited to form the basis of a compiler generating system which is to be implemented on the IBM System/360. It is fairly successful in providing a tool which is superior to assembly code and in meeting the objectives of readability and writability. The language is not as easy to use as some other high-level compiler-writing languages, such as XPL. The execution speed, however, indicates that it may be the superior language in systems and production programming applications.

To make the compiler generating system useful, rigorous debugging of the system should be done. The aforementioned procedures in PROTOCOM , EMIT and LOOKUP, need to be written, along with the nucleus of the procedure SYNTHESIZE. A completely documented and explanatory users' manual is needed to make the system easier to use. It is recommended that these projects be undertaken to make the system a truly useful compiler generating system.

APPENDIX A

SYNTAX ANALYZER LISTING

```
$TITLE   SLR1ANAL
BEGIN

   COMMENT  THE FOLLOWING GLOBAL DATA SEGMENTS CONTAIN ALL OF
            THE NECESSARY ARRAYS THAT ARE REQUIRED TO BUILD THE SLR(1)
            PARSING TABLES.  THE ARRAYS ARE NOT IN A LOGICAL
            ORDER BUT IN AN ORDER THAT WOULD OPTIMIZE THE NUMBER
            OF 4K DATA SEGMENTS. THESE SEGMENTS SHOULD NOT
            BE CHANGED WITHOUT CHANGING THE FIRST OF THE
            MAIN PROGRAM;

   GLCBAL DATA SEGN004 BASE R11;
    ARRAY 4096 BYTE V = ("<SYSTEMGS>_|_");
   CLCSE BASE;
   GLOBAL DATA SEGN005 BASE R12;
   ARRAY 255 INTEGER LOCLENGTH =(#A, #283);
    ARRAY 2048 BYTE PRODARRAY = 0;
   ARRAY 255 BYTE RTPTSIZE = 255(0);
    CLOSE BASE;

    GLOBAL DATA SEGN006 BASE R11;
   ARRAY 255 SHORT INTEGER PRODSTART = 0;
   ARRAY 255 SHORT INTEGER FIRSTPRODFOR;
   ARRAY 255 BYTE ONLEFT = 254(0);
   ARRAY 255 BYTE ONRIGHT = 255(0);
   ARRAY 255 SHORT INTEGER READTOLA = 255(0);
   ARRAY 255 SHORT INTEGER LINEARTOARRAY = 255(0);
   ARRAY 255, SHORT INTEGER TREADSTART = 255(0);
   ARRAY 255 SHORT INTEGER NTREADSTART = 255(0);
   ARRAY 255 BYTE TRDNUM = 255(0);
   ARRAY 255 BYTE NTRDNUM = 255(0);
   CLOSE BASE;

   GLCBAL DATA SEGN007 BASE R10;
   ARRAY 1535 BYTE NTSYMLIST = 1535(0);
   ARRAY 1535 BYTE TSYMLIST = 1535(0);
   ARRAY 255 SHORT INTEGER REDUCESUCC = 255(0);
   ARRAY 127 SHORT INTEGER SUCCSTATE;
   ARRAY 127 SHORT INTEGER FAILSTATE;
   CLCSE BASE;

   GLOBAL DATA SEBN008 BASE R12;
```

```
ARRAY 1535 SHORT INTEGER TSTATELIST = 1535(0);
ARRAY 255 SHORT INTEGER LBSTATE = 255(0);
ARRAY 255 SHORT INTEGER RESUMESTATE;
CLOSE BASE;

GLOBAL DATA SEGN009 BASE R9;
ARRAY 1535 SHORT INTEGER NTSTATELIST = 1535(0);
ARRAY 127 BYTE LASYMNUM;
ARRAY 127 BYTE LBSTART;
ARRAY 127 BYTE LBNUM;
ARRAY 255 SHORT INTEGER BSSTART;
CLOSE BASE;

GLOBAL DATA SEGN010 BASE R8;
ARRAY 2047 BYTE SYMBEFORE = 2047(0);
ARRAY 511 SHORT INTEGER BASISSTACK;
ARRAY 255 BYTE BSSIZE;
ARRAY 127 SHORT INTEGER CONFIGSET;
CLOSE BASE;

GLOBAL DATA SEGN011 BASE R8;
ARRAY 1023 INTEGER LOOKAHEADTABLE = 1023(0);
CLOSE BASE;



EQUATE READLINEAR     SYN 0;
EQUATE READARRAY      SYN 1;
EQUATE REDUCE         SYN 2;
EQUATE LOOKAHEADORD   SYN 3;
EQUATE LOOKAHEADEMPTY SYN 4;
EQUATE LOOKBACK       SYN 5;
EQUATE ERRORSTATE     SYN 6;
EQUATE EXIT           SYN 7;

ARRAY 80 BYTE CBUF;
INTEGER FCBUF SYN CBUF(0);
INTEGER SYMAFTER, CONFIG;
INTEGER NUMREADSTATE;
INTEGER NUMLASTATES, NUMLBSTATES, TRDPTR, NTRDPTR;
BYTE ISLR0, ISLR1, ONESAME;
INTEGER STATE, STATENAME, NUMCONFIGS, SN;
INTEGER SYM, BASISSETSIZE, SETSIZE, LOOKUPSTATE, NEXTSTATE;
INTEGER CLENGTH, LENGTH, LOCATION, LCPT, NUMSYMS = 1, VPT = 12;
INTEGER MINI, MAXI, ERRCOUNT = 0;
BYTE READFLAG, CARDFLG, LEFTSTOP;
ARRAY 255 BYTE CONTROL = 255(0);
ARRAY 132 BYTE WBUF = 131(" ");
```

```
ARRAY 132 BYTE BLANK = 132(" ");
ARRAY 64 BYTE CBCD;
INTEGER NUMTERMINALS, NUMNTS, GOALSYMBOL, I, J;
INTEGER NUMPRODS = 0;
INTEGER REGISTER PAP SYN R10;
ARRAY 1024 BYTE NEWV;
ARRAY 255 BYTE NOTDONE SYN NEWV(0);
ARRAY 255 SHORT INTEGER READTABLE SYN NEWV(256);
BYTE ALLSAME;
BYTE TRUE = #FF;
BYTE FALSE = #00;
ARRAY 4 INTEGER SAVEBASEREGS;
INTEGER NUMREDUCESTATES SYN NUMPRODS;
INTEGER SAVE15;
LONG REAL CONWORK;
INTEGER MARK;
ARRAY 15 SHORT INTEGER BASISSET;
INTEGER TREADNUM, NTREADNUM;
INTEGER BASEREG008;
INTEGER BASE006;
INTEGER BASE005;
INTEGER BASE004;
ARRAY 2 BYTE ARROW ="->";
ARRAY 47 BYTE DASHLINE = 47("-");
ARRAY 25 BYTE TBUF;
INTEGER S;
INTEGER LASTATE;
ARRAY 90 BYTE STARLINE = 90("*");
INTEGER MASK = #80000000;
INTEGER NTSYM, NTSYM2;
BYTE ISVALID, INCL, INTER, CHANGING;
INTEGER PRED, BASE010, BASE011;
INTEGER VLENGTH;

COMMENT  DECLARE TOGGLES TO CONTROL OUTPUT;

BYTE INPUTLIST SYN CONTROL(201);
BYTE GRAMMERLIST SYN CONTROL(199);
BYTE CONFIGLIST SYN CONTROL(195);
BYTE FSMLIST SYN CONTROL(198);
BYTE LASETSLIST SYN CONTROL(211);
BYTE DPDALIST SYN CONTROL(196);
BYTE PUNCHDECK SYN CONTROL(215);
BYTE MOREGRAMMERS;
```

```
FUNCTION SETZONE(8,#96F0);

FUNCTION LOAD(12,#5800);

PROCEDURE MIN(R14);
  BEGIN
    R1 := I; R2 := J;
    IF R1 < R2 THEN MINI := R1
    ELSE MINI := R2;
  END;

PRCCEDURE MAX(R14);
  BEGIN
    R1 := I; R2 := J;
    IF R1 > R2 THEN MAXI := R1
    ELSE MAXI := R2;
  END;

PROCEDURE OUT(R14);
  BEGIN GOTO BAILOUT;
  END;

PROCEDURE ERROR(R14);
  BEGIN
    MVC(9,WBUF,"*** ERROR,");
    RO := @WBUF; WRITE; MVC(100,WBUF,BLANK);
    WRITE; WRITE;
    R1 := ERRCOUNT + 1; ERRCOUNT := R1;
    IF R1 > 15 THEN
    BEGIN
      MVC(40,WBUF,"TOO MANY ERRORS (>15) IN THE CFSM, DPDA, ");
      MVC(34,WBUF(42),"AND/OR LOOK AHEAD SETS.  EXECUTION ");
      MVC(27,WBUF(76),"TERMINATED FOR THIS GRAMMAR.");
      RO := @WBUF; WRITE; MVC(131,WBUF,BLANK);
      OUT;
    END;
  END;

PROCEDURE INCNUMPRODS(R14);
  BEGIN
    R1 := NUMPRODS;
    IF R1 < 255 THEN
    BEGIN
      R1 := R1 + 1; NUMPRODS := R1;
    END ELSE
    BEGIN
      MVC(20,WBUF(15),"TOO MANY PRODUCTIONS.");
```

```
      ERROR;
    END;
  END;

COMMENT  A PROCEDURE TO FIND A TOKEN IN THE ARRAY V;

PROCEDURE FIND(R14);
  BEGIN ARRAY 4 INTEGER SAVEREGS;
    INTEGER SAVE14, SAVE11;
    STM(R1,R4,SAVEREGS);
    SAVE14 := R14; SAVE11 := R11;
    R11 := BASE004;
    MVC(63,CBCD,BLANK); R1 := VPT SHLL 2;
    R3 := LOCLENGTH(R1); R1 := R3 SHRL 6;
    LOCATION := R1; R1 := R3 AND  #3F;
    LENGTH := R1; R2 := LENGTH - 1; CLENGTH := R2;
    FOR R1 := 0 STEP 1 UNTIL R2 DO
    BEGIN
      R3 := LOCATION + R1; IC(R4,V(R3)); STC(R4,CBCD(R1));
    END;
    LM(R1,R4,SAVEREGS);
    R14 := SAVE14; R11 := SAVE11;
  END;  COMMENT END FIND;



GLOBAL PROCEDURE READG(R14);
  BEGIN INTEGER SAVE14;
    INTEGER LP;
    INTEGER REGISTER CP SYN R8;

    PROCEDURE GETCARD(R14);
      BEGIN INTEGER SAVE14;
        SAVE14 := R14;
        WHILE TRUE DO
        BEGIN
          RO := @CBUF; READ;
        IF ¬= THEN
        BEGIN
          RESET(MOREGRAMMERS); IF INPUTLIST THEN PAGE;
          RESET(READFLAG); GOTO ENDGETCARD;
        END;
        IF INPUTLIST THEN
        BEGIN
          MVC(79,WBUF(20),CBUF); RO := @WBUF; WRITE;
          MVC(79,WBUF(20),BLANK);
        END;
```

```
    R1 := R1 - R1; IC(R1,CBUF);
    IF R1 = "@" THEN
    BEGIN
      R1 := FCBUF AND #00FFFFFF;
      IF R1 = "EOG" THEN
      BEGIN
        IF INPUTLIST THEN PAGE;
        RESET(READFLAG); GOTO ENDGETCARD;
      END;
      R1 := R1 SHRL 16 AND #FF; R2 := R2 - R2;
      IC(R2,CONTROL(R1));
      R3 := @CONTROL(R1);
      IF R2 > 0 THEN RESET(B3) ELSE SET(B3);
    END ELSE
    BEGIN
      CLC(79,CBUF,BLANK);
      IF ¬= THEN SET(READFLAG);
      SET(CARDFLG); GOTO ENDGETCARD;
    END;
    END;  COMMENT  END WHILE TRUE;
    ENDGETCARD:
    CP := 0;
    R14 := SAVE14;
  END;

PROCEDURE LOOKUP(R14);
  BEGIN INTEGER SAVE14, SAVE11;
  SAVE14 := R14; SAVE11 := R11;
    R11 := BASE004;
    FOR R1 := 2 STEP 1 UNTIL NUMSYMS DO
    BEGIN
      R2 := R1 SHLL 2; R3 := LOCLENGTH(R2) AND #3F;
      LENGTH := R3; R7 :=CLENGTH + 1;
      IF R7 = LENGTH THEN
      BEGIN
        R5 := LOCLENGTH(R2) SHRL 6;
        R4 := @V(R5) - 1; R2 := @CBCD - 1;
        FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
        BEGIN
          R4 := R4 + 1;  R2 := R2 + 1;  CLC(0,B4,B2);
          IF ¬= THEN GOTO NOTFOUND;
        END;
        LCPT := R1; GOTO ENDLOOKUP;
      END;
      NOTFOUND:
    END;
    R1 := VPT + CLENGTH + 1; VLENGTH := R1;
    IF R1 > 4096 THEN
```

```
      BEGIN
        MVC(31,WBUF(15),"TOO MANY INPUT CHARACTER (>4096)");
        ERROR;
        MVC(37,WBUF,"EXECUTION TERMINATED FOR THIS GRAMMER.");
        RO := @WBUF; WRITE; OUT;
      END;
      R2 := NUMSYMS + 1; NUMSYMS := R2; LCPT := R2;
      R1 := VPT + 1 SHLL 6 + CLENGTH + 1;
      R2 := R2 SHLL 2;
      LOCLENGTH(R2) := R1;
      R1 := VPT;
      FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
      BEGIN
        R1 := R1 + 1;  IC(R5,CBCD(R6)); STC(R5,V(R1));
      END;
      VPT := R1;
      ENDLOOKUP : R14 := SAVE14; R11 := SAVE11;
    END;

  PROCEDURE DEBLANK(R14);
    BEGIN INTEGER SAVE14;
      SAVE14 := R14;
      WHILE R1 = 64  AND CP < 80 DO
      BEGIN
        CP := CP + 1; IC(R1,CBUF(CP)); R1 := R1 AND #FF;
      END;
      R14 := SAVE14;
    END;

  PROCEDURE SCAN(R14);
    BEGIN INTEGER SAVE14;
      SAVE14 := R14;
      IF CP < 80 THEN
      BEGIN
        R1 := R1 - R1; IC(R1,CBUF(CP));
        IF R1 = 64  THEN
        BEGIN
        DEBLANK; R1 := R1 - R1; IC(R1,CBUF(CP));
        END;
      END;
      IF CP < 80 THEN
      BEGIN
        LP := CP;
        IF R1  = "<" THEN SET(LEFTSTOP) ELSE RESET(LEFTSTOP);
        FOR CP := CP + 1 STEP 1 UNTIL 79 DO
        BEGIN
          R1 := R1 - R1; IC(R1,CBUF(CP));
          IF R1 = " " AND LEFTSTOP THEN
```

```
        BEGIN
          R2 := CP - LP;
          IF R2 = 1 THEN
          BEGIN
          CP := CP - 1; GOTO ENDSEARCH;
          END;
        END;
        IF R1 = " " AND ¬LEFTSTOP THEN
        BEGIN
          CP := CP - 1; GOTO ENDSEARCH;
        END;
        IF R1 = ">" AND LEFTSTOP THEN GOTO ENDSEARCH;
      END;
      IF CP = 80 AND LEFTSTOP THEN
      BEGIN
        MVC(17,WBUF(15),"UNMATCHED BRACKET: <");
        ERROR;
      END;
      ENDSEARCH: R2 := CP - LP; CLENGTH := R2;
      R4 := LP - 1;
      FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
      BEGIN
        R4 := R4 + 1;  IC(R3,CBUF(R4)); STC(R3,CBCD(R6));
      END;
      LOOKUP;
      SET(CARDFLG); GOTO ENDSCAN;
    END;
    RESET(CARDFLG);
    ENDSCAN: CP := CP + 1; R14 := SAVE14;
  END;

PROCEDURE FINDGOAL(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14; R1 := 0; GOALSYMBOL := R1;
    FOR R1 := 1 STEP 1 UNTIL NUMSYMS DO
    BEGIN
      R2 := R2 - R2; IC(R2,ONRIGHT(R1));
      IF R2 = 0 THEN
      BEGIN
        R3 := GOALSYMBOL;
        IF R3 = 0 THEN GOALSYMBOL := R1
        ELSE
        BEGIN
          MVC(36,WBUF,"MORE THAN ONE GOAL SYMBOL WAS FOUND:");
          RO := @WBUF; WRITE; MVC(36,WBUF,BLANK);
          VPT := R3; FIND;
          FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
          BEGIN
```

```
      IC(R4,CBCD(R3)); STC(R4,WBUF(R3));
     END;
     MVC(4,WBUF(65),"USED,"); WRITE; MVC(69,WBUF,BLANK);
     VPT := R1; FIND;
     FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
     BEGIN
      IC(R4,CBCD(R3)); STC(R4,WBUF(R3));
     END;
     MVC(7,WBUF(65),"IGNORED."); WRITE; MVC(80,WBUF,BLANK);
    END;
   END;
  END;
  R2 := GOALSYMBOL;
  IF R2 = 0 THEN
  BEGIN
   MVC(33,WBUF,"NO EXPLICIT GOAL SYMBOL WAS FOUND.");
   R0 := @WBUF; WRITE; MVC(33,WBUF,BLANK);
   LH(R3,PRODSTART(2)); IC(R1,PRODARRAY(R3));
   R1 := R1 AND #FF;
   VPT := R1; FIND;
   FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
   BEGIN
    IC(R4,CBCD(R3)); STC(R4,WBUF(R3));
   END;
   MVC(31,WBUF(66),"WILL BE USED AS THE GOAL SYMBOL.");
   WRITE; MVC(100,WBUF,BLANK); WRITE; WRITE;
  END;
  R1 := NUMPRODS + 1 SHLL 1; STH(PAP,PRODSTART(R1));
  R4 := 3; R5 := 0; R1 := R1 SHRL 1;
  STC(R4 ,RTPTSIZE(R1)); STC(R5 ,PRODARRAY(PAP));
  R4 := 1;
  PAP := PAP + 1; STC(R4 ,PRODARRAY(PAP));
  PAP := PAP + 1; R3 := GOALSYMBOL; STC(R3,PRODARRAY(PAP));
  PAP := PAP + 1; STC(R4 ,PRODARRAY(PAP));
  R14 := SAVE14;
 END;  COMMENT  END OF FINDGOAL;

PROCEDURE SORTV(R14);
 BEGIN INTEGER SAVE14;
  ARRAY 255 BYTE INDEX;
  ARRAY 255 BYTE NEWINDEX;
  SAVE14 := R14;
  FOR R2 := 0 STEP 1 UNTIL NUMSYMS DO
  BEGIN
   R1 := R2; STC(R1,INDEX(R2));
  END;
  FOR R2 := 3 STEP 1 UNTIL NUMSYMS DO
  BEGIN
```

```
  R1 := NUMSYMS;
  WHILE R1 >= R2 DO
  BEGIN
    R3 := R1 - 1;
    R4 := R4 - R4; R5 := R5 - R5;
    IC(R4,ONLEFT(R3)); IC(R5,ONLEFT(R1));
    IF R4 = #FF AND  R5 = 0 THEN
    BEGIN
      R4 := @ONLEFT(R3); R5 := @ONLEFT(R1);
      RESET(B4); SET(B5);
      IC(R5,INDEX(R3)); IC(R6,INDEX(R1));
      STC(R6,INDEX(R3)); STC(R5,INDEX(R1));
      R3 := R3 SHLL 2; R1 := R1 SHLL 2;
      R5 := LOCLENGTH(R3); R6 := LOCLENGTH(R1);
      LOCLENGTH(R3) := R6; LOCLENGTH(R1) := R5;
      R3 := R3 SHRL 2; R1 := R1 SHRL 2;
    END;
    R1 := R3;
  END;
END;
FOR R2 := 1 STEP 1 UNTIL NUMSYMS DO
BEGIN
  R1 := R1 - R1; IC(R1,INDEX(R2)); STC(R2,NEWINDEX(R1));
END;
R2 := GOALSYMBOL; R1 := R1 - R1;
IC(R1,NEWINDEX(R2)); GOALSYMBOL := R1;
R1 := NUMPRODS + 1 SHLL 1; R2 := PRODSTART(R1) + 3;
FOR R1 := 1 STEP 1 UNTIL R2 DO
BEGIN
  R3 := R3 - R3; IC(R3,PRODARRAY(R1));
  IC(R4,NEWINDEX(R3)); STC(R4,PRODARRAY(R1));
END;
FOR R1 := 1 STEP 1 UNTIL NUMPRODS DO
BEGIN
  R3 := R1 SHLL 1;
  R2 := PRODSTART(R3); R3 := R3 - 2; R4 := PRODSTART(R3);
  R5 := @PRODARRAY(R2); R6 := @PRODARRAY(R4);
  CLC(0,B5,B6);
  IF ¬= THEN
  BEGIN
    R5 := R5 - R5; IC(R5,PRODARRAY(R2));
    R5 := R5 SHLL 1;
    FIRSTPRODFOR(R5) := R1;
  END;
END;
R1 := 2;
R4 := @ONLEFT(R1);
CLC(0,B4,FALSE);
```

```
  WHILE = DO
  BEGIN
   R1 := R1 + 1; R4 := R4 + 1; CLC(0,B4,FALSE);
  END;
  R1 := R1 - 1;
  NUMTERMINALS := R1;  R2 := NUMSYMS - R1; NUMNTS := R2;
  R14 := SAVE14;
 END;

PROCEDURE PRINTG(R14);
 BEGIN INTEGER SAVE14;
  ARRAY 3 BYTE EQUAL = ("::=");
  SAVE14 := R14;
  MVC(26,WBUF(53),"T H E   V O C A B U L A R Y");
  R0 := @WBUF; WRITE; MVC(26,WBUF(53),BLANK); WRITE; WRITE;
  MVC(30,WBUF(10),"T E R M I N A L   S Y M B O L S");
  MVC(22,WBUF(79),"N O N T E R M I N A L S");
  WRITE; MVC(131,WBUF,BLANK); WRITE; WRITE;
  R1 := NUMTERMINALS; I := R1; R1 := NUMNTS; J := R1; MAX;
  FOR R1 := 1 STEP 1 UNTIL MAXI DO
  BEGIN
   CVD(R1,CONWORK); UNPK(3,7,WBUF(4),CONWORK);
   SETZONE(WBUF(7));
   IF R1 <= NUMTERMINALS THEN
   BEGIN
    VPT := R1; FIND;
    R5 := 9;
    FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
    BEGIN
     R5 := R5 + 1;
     IC(R4,CBCD(R3)); STC(R4,WBUF(R5));
    END;
   END;
   IF R1 <= NUMNTS THEN
   BEGIN
    R2 := R1 + NUMTERMINALS; VPT := R2; FIND;
    R4 := 79;
    FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
    BEGIN
     IC(R5,CBCD(R3)); STC(R5,WBUF(R4)); R4 := R4 + 1;
    END;
   END;
   WRITE; MVC(131,WBUF,BLANK);
  END;
  WRITE; WRITE; MVC(17,WBUF(50),"THE GOAL SYMBOL IS");
  R1 := GOALSYMBOL; VPT := R1; FIND;
  R4 := 79;
  FOR R3 := 0 STEP 1 UNTIL CLENGTH DO
```

```
BEGIN
  IC(R5,CBCD(R3)); STC(R5,WBUF(R4)); R4 := R4 + 1;
END;
WRITE; MVC(70,WBUF(50),BLANK); R1 := MAXI;
IF R1 > 20 THEN PAGE ELSE BEGIN WRITE; WRITE; END;
MVC(28,WBUF(25),"T H E   P R O D U C T I O N S");
WRITE; MVC(28,WBUF(25),BLANK); WRITE; WRITE;
R8 := NUMPRODS;
FOR R1 := 1 STEP 1 UNTIL R8 DO
BEGIN
  R1 := R1 SHLL 1;
  R2 := R2 - R2; R3 := PRODSTART(R1); IC(R2,PRODARRAY(R3));
  R4 := R1 - 2; R5 := PRODSTART(R4); IC(R6,PRODARRAY(R5));
  R6 := R6 AND #FF;
  IF  R2 = R6 THEN
  BEGIN
    R4 := MARK + 2; R3 := #4F; STC(R3,WBUF(R4));
  END ELSE
  BEGIN
    WRITE;
    VPT := R2; FIND; R7 := 10;
    FOR R4 := 0 STEP 1 UNTIL CLENGTH DO
    BEGIN
      IC(R5,CBCD(R4)); STC(R5,WBUF(R7)); R7 := R7 + 1;
    END;
    R7 := R7 + 3; MARK := R7;
    FOR R4 := 0 STEP 1 UNTIL 2 DO
    BEGIN
      IC(R5,EQUAL(R4)); STC(R5,WBUF(R7)); R7 := R7 + 1;
    END;
  END;
  R2 := PRODSTART(R1); R3 := R2 + 1; R4 := R4 - R4;
  R1 := R1 SHRL 1;
  IC(R4,RTPTSIZE(R1)); R2 := R2 + R4; R5 := MARK + 5;
  FOR R3 := R3 STEP 1 UNTIL R2 DO
  BEGIN
    R4 := R4 - R4; IC(R4,PRODARRAY(R3));
    VPT := R4; FIND;
    FOR R4 := 0 STEP 1 UNTIL CLENGTH DO
    BEGIN
      IC(R7,CBCD(R4)); STC(R7,WBUF(R5)); R5 := R5 + 1;
    END;
    R5 := R5 + 2;
  END;
  CVD(R1,CONWORK); UNPK(3,7,WBUF(4),CONWORK);
  SETZONE(WBUF(7));
  WRITE; MVC(131,WBUF,BLANK);
END;
```

```
     PAGE;
     MVC(30,WBUF,"SOME STATISTICS ON THE GRAMMAR:");
     WRITE; MVC(30,WBUF,BLANK); WRITE; WRITE;
     MVC(28,WBUF(5),"NUMBER OF TERMINAL SYMBOLS = ");
     R1 := NUMTERMINALS; CVD(R1,CONWORK);
     UNPK(3,7,WBUF(34),CONWORK); SETZONE(WBUF(37));
     WRITE; MVC(40,WBUF,BLANK);
     MVC(30,WBUF(5),"NUMBER OF NONTERMINAL SYMBOLS = ");
     R1 := NUMNTS; CVD(R1,CONWORK);
     UNPK(3,7,WBUF(37),CONWORK); SETZONE(WBUF(40));
     WRITE; MVC(50,WBUF,BLANK);
     MVC(25,WBUF(11),"TOTAL NUMBER OF SYMBOLS = ");
     R1 := NUMTERMINALS + NUMNTS; CVD(R1,CONWORK);
     UNPK(3,7,WBUF(37),CONWORK); SETZONE(WBUF(40));
     WRITE; MVC(50,WBUF,BLANK); WRITE;
     MVC(23,WBUF(5),"NUMBER OF PRODUCTIONS = ");
     R1 := NUMPRODS; CVD(R1,CONWORK);
     UNPK(3,7,WBUF(29),CONWORK); SETZONE(WBUF(32));
     WRITE; MVC(40,WBUF,BLANK); PAGE;
     R14 := SAVE14;
   END;  COMMENT  END OF PRINTG;


 SAVE14 := R14; CP := 0;
 NUMPRODS := CP; PRODSTART(0) := CP;
 STC(CP,PRODARRAY(0));
 FOR R1 := 0 STEP 1 UNTIL 254 DO
 BEGIN
   STC(CP,ONLEFT(R1)); STC(CP,ONRIGHT(R1));
 END;
 SET(ONLEFT(0)); SET(CNRIGHT(1));
 R1 := 1; NUMSYMS := R1; R10 := R1;
GETCARD;
 WHILE READFLAG DO
 BEGIN
   INCNUMPRODS; CLC(0,CBUF,BLANK);
   IF = THEN
   BEGIN
     R9 := R9 - R9; R2 := NUMPRODS - 1 SHLL 1;
     R3 := PRODSTART(R2); IC(R9,PRODARRAY(R3));
   END ELSE
   BEGIN
     SCAN; R9 := LCPT AND #FF;
   END;
   IC(R2,TRUE); STC(R2,ONLEFT(R9));
   R3 := NUMPRODS SHLL 1; STH(PAP,PRODSTART(R3));
   R3 := _1; R4 := NUMPRODS; STC(R3,RTPTSIZE(R4));
   WHILE ¬CARDFLG DO
```

```
      BEGIN
        STC(R9,PRODARRAY(PAP)); PAP := PAP + 1;
        R4 := NUMPRODS;
        IC(R3,RTPTSIZE(R4)); R3 := R3 + 1; STC(R3,RTPTSIZE(R4));
        SCAN; R9 := LCPT; R2 := #FF; STC(R2,ONRIGHT(R9));
      END;
      R3 := ERRCOUNT;
      IF R3 > 10 THEN
      BEGIN
        MVC(15,WBUF,"TOO MANY ERRORS.");
        MVC(36,WBUF(18),"EXECUTION TERMINATED FOR THIS GRAMMAR");
        RO := @WBUF; WRITE; MVC(52,WBUF,BLANK);
        WHILE READFLAG DO GETCARD;
      END;
      GETCARD;
    END;
    FINDGOAL;
    SORTV;
    R1 := ERRCOUNT;
    IF GRAMMERLIST OR R1 ¬= 0 THEN PRINTG;
    R14 := SAVE14;
  END;  COMMENT  END OF READG;



GLOBAL PROCEDURE COMPUTECFSM(R14);
  BEGIN INTEGER SAVE14; ARRAY 7 INTEGER SAVEREGS;

    PROCEDURE SYMAFTERDOT(R14);
      BEGIN INTEGER SAVE14; ARRAY 4 INTEGER SAVEREGS;
        INTEGER SAVE12;
        SAVE12 := R12; R12 := BASE005;
        SAVE14 := R14; STM(R1,R4,SAVEREGS);
        R1 := CONFIG SHRL 8 AND #FF;
        R3 := R3 - R3;
        R2 := CONFIG AND #FF; IC(R3,RTPTSIZE(R2));
        IF R1 < R3 THEN
        BEGIN
          R2 := R2 SHLL 1; R3 := PRODSTART(R2); R4 := R4 - R4;
          R3 := R3 + R1 + 1; IC(R4,PRODARRAY(R3)); SYMAFTER := R4;
        END ELSE
        BEGIN
          R2 := 0; SYMAFTER := R2;
        END;
        LM(R1,R4,SAVEREGS); R14 := SAVE14; R12 := SAVE12;
      END;  COMMENT  END OF SYMAFTER DOT;

    PROCEDURE INCNUMCONFIGS(R14);
```

```
  BEGIN INTEGER SAVE1, SAVE14;
    SAVE1 := R1; SAVE14 := R14; R1 := NUMCONFIGS + 1;
    IF R1 < 128 THEN NUMCONFIGS := R1
    ELSE BEGIN
      MVC(30,WBUF(15),"THE CONFIGURATION SET FOR STATE");
      R1 := STATE; CVD(R1,CONWORK);  UNPK(3,7,WBUF(47),CONWORK);
      SETZONE(WBUF(50)); MVC(12,WBUF(53),"IS TOO LARGE.");
      ERROR;  SET(CONFIGLIST);
    END;
    R1 := SAVE1; R14 := SAVE14;
  END;  COMMENT END OF INCNUMCONFIGS;

PROCEDURE INCTRDPTR(R14);
  BEGIN INTEGER SAVE1, SAVE14;
    SAVE1 := R1; SAVE14 := R14; R1 := TRDPTR + 1;
    IF R1 < 1536 THEN TRDPTR := R1
    ELSE BEGIN
      MVC(28,WBUF(15),"TOO MANY TERMINAL TRANSITIONS");
      ERROR;
    END;
    R1 := SAVE1; R14 := SAVE14;
  END;  COMMENT  NEND OF INCTRDPTR;

PROCEDURE INCNTRDPTR(R14);
  BEGIN INTEGER SAVE1, SAVE14;
    SAVE1 := R1; SAVE14 := R14; R1 := NTRDPTR + 1;
    IF R1 < 1536 THEN NTRDPTR := R1
    ELSE BEGIN
      MVC(31,WBUF(15),"TOO MANY NONTERMINAL TRANSITIONS");
      ERROR;
    END;
    R14 := SAVE14; R1 := SAVE1;
  END;  COMMENT END OF INCNTRDPTR;

PROCEDURE INCNUMLASTATES(R14);
  BEG'IN  INTEGER SAVE1, SAVE14;
    SAVE1 := R1; SAVE14 := R14; R1 := NUMLASTATES + 1;
    IF R1 < 128 THEN NUMLASTATES := R1
    ELSE BEGIN
      MVC(25,WBUF(15),"TOO MANY LOOK AHEAD STATES");
      ERROR;
  END;
    R1 := SAVE1; R14 := SAVE14;
  END;  COMMENT  END OF INCNUMLASTATES;

PROCEDURE PRINTCS(R14);
  BEGIN INTEGER SAVE14,DOT;
    SAVE14 := R14;
```

```
MVC(30,WBUF,"THE CONFIGURATION SET FOR STATE");
R1 := STATE; CVD(R1,CONWORK);
UNPK(3,7,WBUF(35),CONWORK); SETZONE(WBUF(38));
MVC(2,WBUF(40),"IS:"); R0 := @WBUF; WRITE;
MVC(45,WBUF,BLANK);
FOR R1 := 1 STEP 1 UNTIL NUMCONFIGS DO
BEGIN
  R2 := R1 SHLL 1; R3 := CONFIGSET(R2) AND #FF;
  R3 := R3 SHLL 1; R2 := PRODSTART(R3);
  R3 := R3 - R3; IC(R3,PRODARRAY(R2));
  VPT := R3; FIND; R4 := 10;
  FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
  BEGIN
    IC(R5,CBCD(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
  END;
  R2 := R4 + 3; R4 := R4 + 2;
  FOR R6 := 0 STEP 1 UNTIL 1 DO
  BEGIN
    IC(R5,ARROW(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
  END;
  R1 := R1 SHLL 1; R3 := CONFIGSET(R1) SHRL 8;
  DOT := R3; R1 := R1 SHRL 1;
  IF R3 = 0 THEN
  BEGIN
    R4 := R4 + 1; IC(R5,"."); STC(R5,WBUF(R4));
  END;
  R1 := R1 SHLL 1;
  R5 := R5 - R5; R3 := CONFIGSET(R1) AND #FF;
  R1 := R1 SHRL 1; IC(R5,RTPTSIZE(R3));
  R3 := R3 SHLL 1;
  FOR R2 := 1 STEP 1 UNTIL R5 DO
  BEGIN
    R4 := R4 + 2;
    R6 := PRODSTART(R3) + R2; R7 := R7 - R7;
    IC(R7,PRODARRAY(R6)); VPT := R7; FIND;
    FOR R7 := 0 STEP 1 UNTIL CLENGTH DO
    BEGIN
      IC(R6,CBCD(R7)); STC(R6,WBUF(R4)); R4 := R4 + 1;
    END;
    IF R2 = DOT THEN
    BEGIN
      R4 := R4 + 1; IC(R6,"."); STC(R6,WBUF(R4));
    END;
  END;
  CVD(R1,CONWORK); UNPK(3,7,WBUF(4),CONWORK);
  SETZONE(WBUF(7)); WRITE; MVC(131,WBUF,BLANK);
END;
WRITE; WRITE; R14 := SAVE14;
```

```
  END;  COMMENT  END OF PRINTCS;

PROCEDURE ADDSUCCTOCS(R14);
  BEGIN INTEGER SAVE14; ARRAY 5 INTEGER SAVEREGS;
    SAVE14 := R14; STM(R1,R5,SAVEREGS);
    R2 := SYM SHLL 1; R3 := FIRSTPRODFOR(R2);
    R1 := R3 SHLL 1; R4 := PRODSTART(R1); R5 := R5 - R5;
    IC(R5,PRODARRAY(R4)); R2 := 0;
    WHILE R5 = SYM DO
    BEGIN
      INCNUMCONFIGS; R1 := NUMCONFIGS SHLL 1; R4 := R3 + R2;
      STH(R4,CONFIGSET(R1)); R2 := R2 + 1; R1 := R3 + R2 SHLL 1;
      R4 := PRODSTART(R1); R5 := R5 - R5; IC(R5,PRODARRAY(R4));
    END;
    LM(R1,R5,SAVEREGS); R14 := SAVE14;
  END;  COMMENT  END OF ADDSUCCTOCS;

PROCEDURE SORTCS(R14);
  BEGIN INTEGER TEMP,SAVE14, HOLD;
    SAVE14 := R14; R2 := NUMCONFIGS - 1; TEMP := R2;
    FOR R1 := 0 STEP 1 UNTIL TEMP DO
    BEGIN
      IF R1 = 0 THEN SYM := R1
    ELSE BEGIN
        R1 := R1 SHLL 1; R2 := CONFIGSET(R1); CONFIG := R2;
        SYMAFTERDOT; R2 := SYMAFTER;  SYM := R2;
        R1 := R1 SHRL 1;
      END;
      R3 := R1 + 1 SHLL 1; R4 := CONFIGSET(R3);
      CONFIG := R4; SYMAFTERDOT; R2 := SYM;
      IF R2 ¬= SYMAFTER THEN
      FOR R3 := R1 + 2 STEP 1 UNTIL NUMCONFIGS DO
      BEGIN
        R4 := R3 SHLL 1; R2 := CONFIGSET(R4); CONFIG := R2;
        SYMAFTERDOT; R4 := SYM;
        IF R4 = SYMAFTER THEN
        BEGIN
          HOLD := R2;
          R4 := R3; R5 := R1 + 1;
          WHILE R4 > R5 DO
          BEGIN
            R7 := R4 - 1 SHLL 1; R6 := CONFIGSET(R7);
            R7 := R4 SHLL 1; STH(R6,CONFIGSET(R7)); R4 := R4 -1;
          END;
          R5 := HOLD;
          R7 := R4 SHLL 1; STH(R5,CONFIGSET(R7)); R1 := R4;
        END;
      END;
```

```
    END;
    R14 := SAVE14;
  END;  COMMENT  END OF SORTCS;

PROCEDURE COMPUTECONFIGSET(R14);
  BEGIN INTEGER SAVE14; ARRAY 7 INTEGER SAVEREGS;
    INTEGER SAVE12;
    SAVE12 := R12; R12 := BASE005;
    SAVE14 := R14; STM(R1,R7,SAVEREGS);
    R2 := R2 - R2; R1 := STATE; IC(R2,BSSIZE(R1));
    R2 := R2 - 1; R1 := R1 SHLL 1; R4 := BSSTART(R1);
    FOR R3 := 0 STEP 1 UNTIL R2 DO
    BEGIN
      R5 := R4 + R3 SHLL 1; R6 := BASISSTACK(R5);
      R5 := R3 + 1 SHLL 1; STH(R6,CONFIGSET(R5));
    END;
    NUMCONFIGS := R3; R1 := NUMTERMINALS + 1;
    R2 := NUMTERMINALS + NUMNTS; R3 := @NOTDONE(R1);
    FOR R1 := R1 STEP 1 UNTIL R2 DO
    BEGIN
      SET(B3); R3 := R3 + 1;
    END;
    R1 := 1;
    WHILE R1 <= NUMCONFIGS DO
    BEGIN
      R2 := R1 SHLL 1; R3 := CONFIGSET(R2); CONFIG := R3;
      SYMAFTERDOT; R2 := SYMAFTER; R3 := R3 - R3;
      R4 := R4 - R4; IC(R3,ONLEFT(R2)); IC(R4,NOTDONE(R2));
      IF R3 ¬= 0 AND R4 ¬= 0 AND R2 ¬= 0 THEN
      BEGIN
        SYM := R2; ADDSUCCTOCS; R5 := @NOTDONE(R2); RESET(B5);
      END;
      R1 := R1 + 1;
    END;
    R1 := NUMCONFIGS;
    IF R1 > 1 THEN SORTCS;
    IF CONFIGLIST THEN PRINTCS;
    R14 := SAVE14; LM(R1,R7,SAVEREGS); R12 := SAVE12;
  END;  COMMENT  END OF COMPUTECONFIGSET;

PROCEDURE INCBASISSETSIZE(R14);
  BEGIN INTEGER SAVE1,SAVE14;
    SAVE1 := R1; SAVE14 := R14; R1 := BASISSETSIZE + 1;
    IF R1 < 16 THEN BASISSETSIZE := R1
    ELSE BEGIN
      MVC(22,WBUF(15),"THE BASIS SET FOR STATE");
      R1 := STATE;CVD(R1,CONWORK); UNPK(3,7,WBUF(39),CONWORK);
      SETZONE(WBUF(42)); MVC(9,WBUF(45),"IS TOO BIG");
```

```
      ERROR;
    END;
    R1 := SAVE1; R14 := SAVE14;
  END;

PROCEDURE LOOKUPREAD(R14);
  BEGIN INTEGER SAVE14; ARRAY 7 INTEGER SAVEREGS;
    SAVE14 := R14; STM(R1,R7,SAVEREGS);
    FOR R1 := 1 STEP 1 UNTIL NUMREADSTATES DO
    BEGIN
      R2 := R2 - R2; IC(R2,BSSIZE(R1));
      IF R2 = SETSIZE THEN
      BEGIN
        SET(ALLSAME);
        R2 := R1 SHLL 1; R3 := BSSTART(R2);
        R4 := R3 + SETSIZE - 1;
        FOR R3 := R3 STEP 1 UNTIL R4 DO
        BEGIN
          RESET(ONESAME);
          R2 := R3 SHLL 1; R6 := BASISSTACK(R2);
          FOR R2 := 1 STEP 1 UNTIL SETSIZE DO
          BEGIN
            R7 := R2 SHLL 1;
            IF R6 = BASISSET(R7) THEN SET(ONESAME);
          END;
          NC(0,ALLSAME,ONESAME);
        END;
        IF ALLSAME THEN
        BEGIN
          LASTATE := R1; GOTO ENDLOOKUPREAD;
        END;
      END;
    END;
    LASTATE := R1; R1 := NUMREADSTATES + 1;
    IF R1 < 256 THEN NUMREADSTATES := R1
    ELSE BEGIN
      MVC(19,WBUF(15),"TOO MANY READ STATES");
      ERROR;
    END;
    R1 := LASTATE - 1; R3 := R1 SHLL 1; R2 := R2 - R2;
    IC(R2,BSSIZE(R1)); R2 := R2 + BSSTART(R3); R3 := R3 + 2;
    BSSTART(R3) := R2; R3 := R3 SHRL 1; R1 := SETSIZE;
    STC(R1,BSSIZE(R3)); R1 := R1 + R2;
    IF R1 > 511 THEN
    BEGIN
      MVC(32,WBUF(15),"CFSM(THE SET OF BASIS SET)IS TOO LARGE");
      ERROR; R1 := 512 - SETSIZE; R2 := LASTATE SHLL 1;
      STH(R1,BSSTART(R2));
```

```
    END;
    R2 := LASTATE SHLL 1; R1 := BSSTART(R2) - 1 SHLL 1;
    R4 := SETSIZE SHLL 1;
    FOR R3 := 2 STEP 2 UNTIL R4 DO
    BEGIN
      R6 := R1 + R3;
      R5 := BASISSET(R3); STH(R5,BASISSTACK(R6));
    END;
    ENDLOOKUPREAD:
    LM(R1,R7,SAVEREGS); R14 := SAVE14;
  END;  COMMENT  END LOOKUPREAD;

PROCEDURE ADDREADXITION(R14);
  BEGIN  ARRAY 4 INTEGER SAVEREGS;
    STM(R1,R4,SAVEREGS); R1 := SYM; R2 := NEXTSTATE;
    R3 := @ONLEFT(R1);
    CLC(0,TRUE,B3);
    IF = THEN
    BEGIN
      R4 := NTREADNUM + 1; NTREADNUM := R4;
    END ELSE
    BEGIN
      R4 := TREADNUM + 1; TREADNUM := R4;
    END;
    R1 := R1 SHLL 1; STH(R2,READTABLE(R1));
    LM(R1,R4,SAVEREGS);
  END;  COMMENT  END OF ADDREADXITION;

PROCEDURE ENCODEREAD(R14);
  BEGIN INTEGER SAVE14; ARRAY 7 INTEGER SAVEREGS;
    INTEGER SAVE12;
    SAVE14 := R14; STM(R1,R7,SAVEREGS);
    SAVE12 := R12; R12 := BASEREG008;
    R1 := TREADNUM; R3 := R1 * 3;
    IF R3 < NUMTERMINALS AND R1 < 16 THEN
    BEGIN
      R3 := TRDPTR; R2 := STATE SHLL 1;
      STH(R3,TREADSTART(R2)); R2 := R2 SHRL 1;
      STC(R1,TRDNUM(R2));
      FOR R4 := 1 STEP 1 UNTIL NUMTERMINALS DO
      BEGIN
        R2 := R4 SHLL 1; R5 := READTABLE(R2);
        IF R5 ¬= #6FF THEN
        BEGIN
          R3 := TRDPTR;
          STC(R4,TSYMLIST(R3)); R3 := R3 SHLL 1;
          STH(R5,TSTATELIST(R3)); INCTRDPTR;
        END;
```

```
      END;
      R4 := ERRORSTATE SHLL 8 OR 255;
      R3 := TRDPTR SHLL 1; STH(R4,TSTATELIST(R3));
      INCTRDPTR;
    END ELSE
    BEGIN
      R2 := STATE; R1 := READARRAY SHLL 8 OR R2;
      R3 := R2 SHLL 1; STH(R1,LINEARTOARRAY(R3)); R4 := TRDPTR;
      STH(R4,TREADSTART(R3)); R4 := NUMTERMINALS + 1;
      STC(R4,TRDNUM(R2));
      FOR R4 := 0 STEP 1 UNTIL NUMTERMINALS DO
      BEGIN
        R2 := TRDPTR + R4; STC(R4,TSYMLIST(R2));
        R3 := R4 SHLL 1; R5 := READTABLE(R3);  R2 := R2 SHLL 1;
        STH(R5,TSTATELIST(R2));
      END;
      R2 := TRDPTR + NUMTERMINALS; TRDPTR := R2;
      INCTRDPTR;
    END;
    R2 := NTRDPTR; R1 := STATE SHLL 1;
    STH(R2,NTREADSTART(R1)); R1 := R1 SHRL 1;
    R5 := NTREADNUM; STC(R5,NTRDNUM(R1));
    R5 := NUMTERMINALS + NUMNTS;
    FOR R1 := NUMTERMINALS + 1 STEP 1 UNTIL R5 DO
    BEGIN
      R2 := NTRDPTR; R3 := R1 SHLL 1; R4 := READTABLE(R3);
      IF R4 ¬= #6FF THEN
      BEGIN
        STC(R1,NTSYMLIST(R2)); R2 := R2 SHLL 1;
        STH(R4,NTSTATELIST(R2)); INCNTRDPTR;
      END;
    END;
    LM(R1,R7,SAVEREGS); R14 := SAVE14; R12 := SAVE12;
  END;  COMMENT  END OF ENCODEREAD;

PROCEDURE ADDSUCCESSORSTOBS(R14);
  BEGIN INTEGER SAVE14; ARRAY 7 INTEGER SAVEREGS;
    INTEGER SAVE12;
    SAVE14 := R14; STM(R1,R7,SAVEREGS);
    SAVE12 := R12; R12 := BASE005;
    R1 := 0; BASISSETSIZE := R1;
    FOR R2 := 0 STEP 2 UNTIL 28 DO STH(R1,BASISSET(R2));
    R1 := 1;
    R2 := R1 SHLL 1; R3 := CONFIGSET(R2); CONFIG := R3;
    SYMAFTERDOT; R2 := SYMAFTER;
    WHILE R1 <= NUMCONFIGS AND R2 = 0 DO
    BEGIN
      R3 := R3 AND #FF; R4 := R4 - R4; IC(R4,RTPTSIZE(R3));
```

```
  IF R4 ¬= 0 THEN R5 := LOOKAHEADORD
  ELSE R5 := LOOKAHEADEMPTY;
  R5 := R5 SHLL 8 OR NUMLASTATES;
  IF R1 = 1 THEN
  BEGIN
    R2 := STATE SHLL 1;
    RESET(ISLRO); STH(R5,READTOLA(R2));
  END;
  R2 := NUMLASTATES; R6 := R3 SHLL 1;
  R7 := PRODSTART(R6); IC(R6,PRODARRAY(R7));
  STC(R6,LASYMNUM(R2));
  R6 := REDUCE SHLL 8 OR R3; R2 := R2 SHLL 1;
  STH(R6,SUCCSTATE(R2));
  IF R1 > 1 THEN
  BEGIN
    R2 := R2 - 2;  STH(R5,FAILSTATE(R2));
  END;
  R2 := STATE; IC(R4,SYMBEFORE(R2)); STC(R4,SYMBEFORE(R5));
  INCNUMLASTATES; R1 := R1 + 1;
  R2 := R1 SHLL 1; R3 := CONFIGSET(R2); CONFIG := R3;
  SYMAFTERDOT; R2 := SYMAFTER;
END;
IF R1 > 1 THEN
BEGIN
IF R1 > NUMCONFIGS THEN
BEGIN
  R2 := NUMLASTATES - NUMCONFIGS;
  R3 := ERRORSTATE SHLL 8 OR R2;
  R2 := NUMLASTATES - 1 SHLL 1; STH(R3,FAILSTATE(R2));
END ELSE
BEGIN
  R3 := STATE; R2 := NUMLASTATES - 1 SHLL 1;
  STH(R3,FAILSTATE(R2));
END;
END;
R2 := NUMTERMINALS + NUMNTS SHLL 1; R3 := #6FF;
FOR R4 := 0 STEP 2 UNTIL R2 DO STH(R3,READTABLE(R4));
R4 := 0;TREADNUM := R4; NTREADNUM := R4;
WHILE R1 <= NUMCONFIGS DO
BEGIN
  R4 := 0; BASISSETSIZE := R4;
  R2 := R1 SHLL 1; R3 := CONFIGSET(R2); CONFIG := R3;
  SYMAFTERDOT; R5 := SYMAFTER; SYM := R5;
  WHILE R5 = SYMAFTER AND R1 <= NUMCONFIGS DO
  BEGIN
    INCBASISSETSIZE; R2 := BASISSETSIZE SHLL 1;
    R4 := R3 + 256; STH(R4,BASISSET(R2)); R1 := R1 + 1;
    R2 := R1 SHLL 1; R3 := CONFIGSET(R2); CONFIG := R3;
```

```
        SYMAFTERDOT;
      END;
      R2 := BASISSETSIZE; R3 := BASISSET(2); CONFIG := R3;
      SYMAFTERDOT; R3 := SYMAFTER;
      IF R2 = 1 AND R3 = 0 THEN
      BEGIN
        R3 := BASISSET(2) AND #FF;
        R5 := REDUCE SHLL 8 OR R3;
      END ELSE
      BEGIN
        SETSIZE := R2; LOOKUPREAD; R5 := LASTATE;
      END;
      NEXTSTATE := R5; ADDREADXITION;
      R2 := SYM; STC(R2,SYMBEFORE(R5));
      IF R1 > NUMCONFIGS THEN ENCODEREAD;
    END;
    R14 := SAVE14; LM(R1,R7,SAVEREGS); R12 := SAVE12;
  END;  COMMENT  END OF ADDSUCCESSORSTOBS;


PROCEDURE CONVERT(R14);
  BEGIN ARRAY 3 INTEGER SAVEREGS; INTEGER SAVE14;
    STM(R1,R3,SAVEREGS); SAVE14 := R14;
    R1 := STATENAME SHLL 1; R2 := READTOLA(R1);
    IF R2 > 767 THEN SN := R2
    ELSE BEGIN
      R2 := R2 SHLL 1; R3 := LINEARTOARRAY(R2); SN := R3;
    END;
    LM(R1,R3,SAVEREGS); R14 := SAVE14;
  END;  COMMENT  END OF CONVERT;

STM(R1,R7,SAVEREGS); SAVE14 := R14;
INCNUMPRODS;
R1 := NUMPRODS SHLL 1; R2 := PRODSTART(R1);
R1 := 0; NUMREADSTATE := R1; TRDPTR := R1; NTRDPTR := R1;
STC(R1,PRODARRAY(R2));
R12 := BASEREG008;
NUMLASTATES := R1; R2 := 1; R3 := NUMREADSTATES;
STC(R2,BSSIZE(R3)); R3 := R3 SHLL 1; STH(R1,BSSTART(R3));
R2 := NUMPRODS;          STH(R2,BASISSTACK(0));
WHILE R1 <= NUMREADSTATES DO
BEGIN
  STATE := R1; R2 := R1 SHLL 1; STH(R1,READTOLA(R2));
  STH(R1,LINEARTOARRAY(R2)); COMPUTECONFIGSET;
  ADDSUCCESSORSTOBS; R1 := R1 + 1;
END;
R2 := NUMPRODS SHLL 1; R3 := EXIT SHLL 8;
STH(R3,REDUCESUCC(R2));
```

```
FOR R1 := 0 STEP 1 UNTIL NUMREADSTATES DO
BEGIN
  R2 := R1 SHLL 1; R3 := TREADSTART(R2);
  R4 := R4 - R4; IC(R4,TRDNUM(R1));
  R4 := R4 + R3 - 1 SHLL 1;
  FOR R3 := R3 SHLL 1 STEP 2 UNTIL R4 DO
  BEGIN
    R2 := TSTATELIST(R3);
    IF R2 <= 255 THEN
    BEGIN
      STATENAME := R2; CONVERT; R2 := SN;
      STH(R2,TSTATELIST(R3));
    END;
  END;
  R2 := R1 SHLL 1; R3 := NTREADSTART(R2);
  R4 := R4 - R4; IC(R4,NTRDNUM(R1));
  R4 := R4 + R3 - 1 SHLL 1;
  FOR R3 := R3 SHLL 1 STEP 2 UNTIL R4 DO
  BEGIN
    R2 := NTSTATELIST(R3);
    IF R2 <= 255 THEN
    BEGIN
      STATENAME := R2; CGNVERT; R2 := SN;
      STH(R2,NTSTATELIST(R3));
    END;
  END;
END;
R7 := NUMLASTATES - 1 SHLL 1;
FOR R1 := 0 STEP 2 UNTIL R7 DO
BEGIN
  R2 := FAILSTATE(R1);
  IF R2 < 256 THEN
  BEGIN
    R2 := R2 SHLL 1;
    R3 := LINEARTOARRAY(R2); STH(R3,FAILSTATE(R1));
  END;
END;
LM(R1,R7,SAVEREGS); R14 := SAVE14; R12 := BASE005;
END;  COMMENT  END OF COMPUTECFSM;



PROCEDURE CONVSTATE(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14; MVC(24,TBUF,BLANK);
    R14 := S SHRL 8 + 1;
    CASE R14 OF
    BEGIN
```

```
      MVC(10,TBUF,"READ LINEAR");
      MVC(9,TBUF,"READ ARRAY");
      MVC(5,TBUF,"REDUCE");
      MVC(13,TBUF,"LOOK AHEAD ORD");
      MVC(15,TBUF,"LOOK AHEAD EMPTY");
      MVC(8,TBUF,"LOOK BACK");
      MVC(10,TBUF,"ERROR (FROM");
      MVC(3,TBUF,"EXIT");
    END;
    R14 := S AND #FF;
    CVD(R14,CONWORK); UNPK(3,7,TBUF(18),CONWORK);
    SETZONE(TBUF(21));
    R14 := S SHRL 8 + 1;
    IF R14 = 7 THEN MVC(0,TBUF(23),")");
    R14 := SAVE14;
  END;  COMMENT  END OF CONVSTATE;

PROCEDURE PRINTCFSM(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14;
    MVC(37,WBUF,"THE CFSM FOR THE GRAMMAR IS AS FOLLOWS");
    RO := @WBUF; WRITE; MVC(37,WBUF,BLANK);  WRITE; WRITE;
    FOR R1 := 0 STEP 1 UNTIL NUMREADSTATES DO
    BEGIN
      R2 := R1 SHLL 1; R3 := READTOLA(R2);
      IF R3 ¬= R1 THEN
      BEGIN
        MVC(10,WBUF,"READ STATE "); CVD(R1,CONWORK);
        UNPK(3,7,WBUF(13),CONWORK); SETZONE(WBUF(16));
        MVC(12,WBUF(18),"IS INADEQUATE");
        MVC(37,WBUF(34),"THE FOLLOWING LOOK-AHEAD IS NECESSARY.");
        WRITE; MVC(72,WBUF,BLANK); WRITE; WRITE; R4 := R3 SHRL 8;
        WHILE R4 = LOOKAHEADORD OR R4 = LOOKAHEADEMPTY DO
        BEGIN
          S := R3; CONVSTATE; MVC(24,WBUF,TBUF);
          R3 := R3 AND #FF;
          MVC(0,WBUF(24),":"); MVC(11,WBUF(32),"LA SYM NUM =");
          R4 := R4 - R4; IC(R4,LASYMNUM(R3)); R4 := R4 + 1;
          CVD(R4,CONWORK); UNPK(3,7,WBUF(45),CONWORK);
          SETZONE(WBUF(48)); MVC(0,WBUF(49),")");
          WRITE; MVC(49,WBUF,BLANK); WRITE;
          MVC(8,WBUF,"LA SET OF"); R4 := R3 SHLL 1;
          R5 := SUCCSTATE(R4) AND #FF; R5 := R5 SHLL 1;
          R4 := PRODSTART(R5); R5 := R5 - R5; IC(R5,PRODARRAY(R4));
          VPT := R5; FIND; R4 := 36 - CLENGTH;
          FOR R7 := 0 STEP 1 UNTIL CLENGTH DO
          BEGIN
            IC(R5,CBCD(R7)); STC(R5,WBUF(R4)); R4 := R4 + 1;
```

```
    END;
    R4 := 39;
    FOR R7 := 0 STEP 1 UNTIL 1 DO
    BEGIN
      IC(R5,ARROW(R7)); STC(R5,WBUF(R4)); R4 := R4 + 1;
    END;
    R4 := R3 SHLL 1; R5 := SUCCSTATE(R4);
    S := R5; CONVSTATE; MVC(24,WBUF(42),TBUF);
    WRITE; MVC(67,WBUF,BLANK);
    R3 := FAILSTATE(R4); S := R3; CONVSTATE;
    MVC(24,WBUF(42),TBUF);
    MVC(10,WBUF(30),"DEFAULT  ->"); WRITE;
    MVC(67,WBUF,BLANK); WRITE; WRITE;
    R4 := R3 SHRL 8;
  END;
END;
R2 := R1 SHLL 1; R3 := LINEARTOARRAY(R2);
S := R3; CONVSTATE; MVC(4,WBUF,"STATE");
MVC(24,WBUF(6),TBUF);  MVC(0,WBUF(28),":");
MVC(18,WBUF(33),"ACCESSING SYMBOL IS"); R4 := R4 - R4;
IC(R4,SYMBEFORE(R1)); VPT := R4; FIND; R4 := 54;
FOR R2 := 0 STEP 1 UNTIL CLENGTH DO
BEGIN
  IC(R5,CBCD(R2)); STC(R5,WBUF(R4)); R4 := R4 + 1;
END;
IC(R5,")"); STC(R5,WBUF(R4)); WRITE;
MVC(131,WBUF,BLANK); WRITE;
R2 := R1 SHLL 1; R3 := TREADSTART(R2); R7 := R7 - R7;
IC(R7,TRDNUM(R1)); R7 := R7 + R3 - 1;
FOR R3 := R3 STEP 1 UNTIL R7 DO
BEGIN
  R2 := R2 - R2; IC(R2,TSYMLIST(R3)); VPT := R2; FIND;
  R4 := 36 - CLENGTH;
  FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
  BEGIN
    IC(R5,CBCD(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
  END;
  R4 := 39;
  FOR R6 := 0 STEP 1 UNTIL 1 DO
  BEGIN
    IC(R5,ARROW(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
  END;
  R12 := BASEREG008;
  R2 := R3 SHLL 1; R4 := TSTATELIST(R2); S := R4;
  R12 := BASE005;
  CONVSTATE; MVC(24,WBUF(42),TBUF);
  WRITE; MVC(67,WBUF,BLANK);
END;
```

```
      MVC(46,WBUF(12),DASHLINE); WRITE; MVC(60,WBUF,BLANK);
      R2 := R1 SHLL 1; R3 := NTREADSTART(R2); R7 := R7 - R7;
      IC(R7,NTRDNUM(R1)); R7 := R7 + R3 - 1;
      FOR R3 := R3 STEP 1 UNTIL R7 DO
      BEGIN
        R2 := R2 - R2; IC(R2,NTSYMLIST(R3)); VPT := R2; FIND;
        R4 := 36 - CLENGTH;
        FOR R6 := 0 STEP 1 UNTIL CLENGTH DO
        BEGIN
          IC(R5,CBCD(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
        END;
        R4 := 39;
        FOR R6 := 0 STEP 1 UNTIL 1 DO
        BEGIN
          IC(R5,ARROW(R6)); STC(R5,WBUF(R4)); R4 := R4 + 1;
        END;
        R2 := R3 SHLL 1; R4 := NTSTATELIST(R2);
        S := R4; CONVSTATE; MVC(24,WBUF(42),TBUF); WRITE;
        MVC(67,WBUF,BLANK);
      END;
      WRITE; WRITE;
    ENC;
    PAGE;
    R14 := SAVE14;
  END;  COMMENT  END OF PRINTCFSM;

    PROCEDURE ISVALIDLA(R14);
      BEGIN ARRAY 3 INTEGER SAVEREGS;
        STM(R1,R3,SAVEREGS);
        R2 := SYM SHRL 5;
        R1 := NTSYM - NUMTERMINALS SHLL 3 OR R2 SHLL 2;
        R2 := SYM AND #1F; R3 := MASK SHRL R2;
        R2 := LOOKAHEADTABLE(R1) AND R3;
        IF R2 = 0 THEN RESET(ISVALID) ELSE SET(ISVALID);
        LM(R1,R3,SAVEREGS);
      END;  COMMENT  END OF ISVALIDLA;

GLOBAL PROCEDURE LOOKAHEADANDDPDA(R14);
  BEGIN INTEGER SAVE14;
    BYTE ERRORST;

    PROCEDURE VALIDATELAXITION(R14);
      BEGIN ARRAY 4 INTEGER SAVEREGS;
        STM(R1,R4,SAVEREGS);
        R1 := NTSYM - NUMTERMINALS SHLL 3; R2 := SYM SHRL 5;
        R1 := R1 OR R2 SHLL 2;
        R2 := LOOKAHEADTABLE(R1); R4 := SYM AND #1F;
        R3 := MASK SHRL R4; R2 := R2 OR R3;
```

```
    LOOKAHEADTABLE(R1) := R2;
    LM(R1,R4,SAVEREGS);
  END;  COMMENT  END OF VALIDLAXITIIN;

PROCEDURE INTERSECT(R14);
  BEGIN ARRAY 4 INTEGER SAVEREGS;
    STM(R1,R4,SAVEREGS);
    R1 := NTSYM - NUMTERMINALS SHLL 5;
    R2 := NTSYM2 - NUMTERMINALS SHLL 5;
    RESET(INTER);
    FOR R3 := 0 STEP 1 UNTIL 3 DO
    BEGIN
      R4 := LOOKAHEADTABLE(R1) AND LOOKAHEADTABLE(R2);
      IF R4 ¬= 0 THEN OI(#FF,INTER);
      R1 := R1 + 4; R2 := R2 + 4;
    END;
    LM(R1,R4,SAVEREGS);
  END;  COMMENT  END OF INTERSECT;

PROCEDURE INCLUDED(R14);
  BEGIN  ARRAY 5 INTEGER  SAVEREGS;
    STM(R1,R5,SAVEREGS);
    R1 := SYM - NUMTERMINALS SHLL 5;
    R2 := PRED - NUMTERMINALS SHLL 5;
    SET(INCL);
    FOR R3 := 0 STEP 1 UNTIL 3 DO
    BEGIN
      R4 := LOOKAHEADTABLE(R2);
      R5 := LOOKAHEADTABLE(R1) AND R4;
      IF R5 = R4 THEN NI(#FF,INCL)
      ELSE NI(#00,INCL);
      R2 := R2 + 4; R1 := R1 + 4;
    END;
    LM(R1,R5,SAVEREGS);
  END;  COMMENT  END OF INCLUDED;

PROCEDURE INCLUDE(R14);
  BEGIN  ARRAY 4 INTEGER SAVEREGS;
    STM(R1,R4,SAVEREGS);
    R1 := SYM - NUMTERMINALS SHLL 5;
    R2 := PRED - NUMTERMINALS SHLL 5;
    FOR R3 := 0 STEP 1 UNTIL 3 DO
    BEGIN
      R4 := LOOKAHEADTABLE(R1) OR LOOKAHEADTABLE(R2);
      LOOKAHEADTABLE(R1) := R4; R1 := R1 + 4;
      R2 := R2 + 4;
    END;
    LM(R1,R4,SAVEREGS);
```

```
END;  COMMENT  END OF INCLUDE;

PROCEDURE NOTSLR1(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14;
    RESET(ISLR1); MVC(89,WBUF,STARLINE);
    RO := @WBUF; WRITE; MVC(90,WBUF,BLANK); WRITE; WRITE;
    CONVSTATE; MVC(25,WBUF,"THE GRAMMAR IS NOT SLR(1):");
    R14 := STATE; CVD(R14,CONWORK);
    UNPK(3,7,WBUF(27),CONWORK); SETZONE(WBUF(30));
    MVC(23,WBUF(32),"INTERSECTS THAT OF STATE");
    MVC(24,WBUF(56),TBUF); MVC(0,WBUF(81),".");
    WRITE; MVC(82,WBUF,BLANK); WRITE; WRITE;
    MVC(89,WBUF,STARLINE); WRITE; MVC(89,WBUF,BLANK); WRITE;
    R14 := SAVE14;
  END;  COMMENT  END OF NOTSLR1;

PROCEDURE PRINTSLR1LASETS(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14; R1 := NUMNTS; IF R1 > 18 THEN PAGE;
    R1 := NUMTERMINALS;
    IF R1 > 50 THEN
    BEGIN
      WRITE; MVC(17,WBUF,"NUMTERMINALS > 50,"); WRITE;
      MVC(33,WBUF,"ONLY THE FIRST 50 WILL BE PRINTED.");
      WRITE; MVC(33,WBUF,BLANK); WRITE;
    END;
    MVC(36,WBUF(11),"T H E   L G O K - A H E A D   S E T S");
    RO := @WBUF; WRITE; MVC(36,WBUF(11),BLANK); WRITE; WRITE;
    MVC(31,WBUF,"NONTERMINAL      TERMINAL SYMBOLS");
    WRITE; MVC(31,WBUF,BLANK);
    MVC(5,WBUF(3),"SYMBOL"); R2 := 11;
    FOR R4 := 1 STEP 1 UNTIL 2 DO
    BEGIN
      FOR R1 := R4 STEP 2 UNTIL NUMTERMINALS DO
      BEGIN
        IF R1 < 10 THEN
        BEGIN
          R2 := R2 + 4; CVD(R1,CONWORK); R3 := @WBUF(R2);
          UNPK(0,7,B3,CONWORK); SETZONE(B3);
        END ELSE
        BEGIN
          R2 := R2 + 4; CVD(R1,CONWORK); R3 := @WBUF(R2)- 1;
          UNPK(1,7,B3,CONWORK); R3 := R3 + 1;
          SETZONE(B3);
        END;
      END;
      RO := @WBUF; WRITE; MVC(131,WBUF,BLANK);
```

```
         R2 := 13;
      END;
      IC(R1,DASHLINE); R2 := NUMTERMINALS SHLL 1 + 13;
      FOR R3 := 13 STEP 1 UNTIL R2 DO STC(R1,WBUF(R3));
      WRITE; MVC(131,WBUF,BLANK);
      R2 := NUMTERMINALS + NUMNTS;
      FOR R1 := NUMTERMINALS + 1 STEP 1 UNTIL R2 DO
      BEGIN
         VPT := R1; FIND; R3 := 11 - CLENGTH;
         IF R3 < 0 THEN
         BEGIN
            R4 := CLENGTH + R3; R3 := 0;
         END ELSE R4 := CLENGTH;
         FOR R5 := 0 STEP 1 UNTIL R4 DO
         BEGIN
            IC(R6,CBCD(R5)); STC(R6,WBUF(R3)); R3 := R3 + 1;
         END;
         MVC(0,WBUF(13),"|"); R3 := 15; R4 := #F1;
         R7 := NUMTERMINALS; IF R7 > 50 THEN R7 := 50;
         FOR R5 := 1 STEP 1 UNTIL R7 DO
         BEGIN
            NTSYM := R1; SYM := R5; ISVALIDLA;
            IF ISVALID THEN STC(R4,WBUF(R3)); R3 := R3 + 2;
         END;
         WRITE; MVC(131,WBUF,BLANK);
      END;
      PAGE; R14 := SAVE14;
   END;  COMMENT  END OF PRINTSLR1LASETS;


R8 := BASE011; SAVE14 := R14;
FOR R1 := 1 STEP 1 UNTIL NUMREADSTATES DO
BEGIN
   R8 := BASE010; R2 := R2 - R2; IC(R2,SYMBEFORE(R1));
   R8 := BASE011; NTSYM := R2;
   R3 := @ONLEFT(R2);
   CLC(0,B3,TRUE);
   IF = THEN
   BEGIN
      R2 := R1 SHLL 1; R3 := TREADSTART(R2);
      R2 := R2 - R2; IC(R2,TRDNUM(R1)); R2 := R2 + R3 SHLL 1;
      FOR R3 := R3 SHLL 1 STEP 2 UNTIL R2 DO
      BEGIN
         R12 := BASEREG008; R4 := TSTATELIST(R3); R12 := BASE005;
         IF R4 ¬= #6FF THEN
         BEGIN
            R4 := R3 SHRL 1; R5 := R5 - R5;
            IC(R5,TSYMLIST(R4)); SYM := R5; VALIDATELAXITION;
```

```
      END;
    END;
  END;
END;
R2 := NUMPRODS - 1;
FOR R1 := 1 STEP 1 UNTIL R2 DO
BEGIN
  R3 := R3 - R3; IC(R3,RTPTSIZE(R1));
  IF R3 ¬= 0 THEN
  BEGIN
    R4 := R1 SHLL 1; R5 := PRODSTART(R4); R5 := R5 + R3;
    R4 := R4 - R4; IC(R4,PRODARRAY(R5)); R3 := @ONLEFT(R4);
    CLC(0,B3,TRUE);
    IF = THEN
    BEGIN
      NTSYM := R4; R1 := R1 SHLL 1; R4 := PRODSTART(R1);
      R5 := R5 - R5; IC(R5,PRODARRAY(R4)); SYM := R5;
      VALIDATELAXITION; R1 := R1 SHRL 1;
    END;
  END;
END;
SET(CHANGING);
WHILE CHANGING DO
BEGIN
  RESET(CHANGING); R2 := NUMTERMINALS + NUMNTS;
  FOR R1 := NUMTERMINALS + 1 STEP 1 UNTIL R2 DO
  BEGIN
    FOR R3 := NUMTERMINALS + 1 STEP 1 UNTIL R2 DO
    BEGIN
      NTSYM := R1; SYM := R3; ISVALIDLA;
      IF ISVALID THEN
      BEGIN
        SYM := R1; PRED := R3; INCLUDED;
        IF ¬INCL THEN
        BEGIN
          SET(CHANGING); INCLUDE;
        END;
      END;
    END;
  END;
END;
R2 := NUMLASTATES - 1;
FOR R1 := 0 STEP 1 UNTIL R2 DO
BEGIN
  STATE := R1; R3 := R1 SHLL 1; R4 := FAILSTATE(R3); S:= R4;
  R5 := SUCCSTATE(R3) AND #FF SHLL 1;
  R6 := PRODSTART(R5); R7 := R7 - R7; IC(R7,PRODARRAY(R6));
  NTSYM := R7; R3 := R4 AND #FF SHLL 1;
```

```
    R5 := SUCCSTATE(R3) AND #FF SHLL 1;
    R6 := PRODSTART(R5); R7 := R7 - R7; IC(R7,PRODARRAY(R6));
    NTSYM2 := R7; INTERSECT;
    WHILE R4 > 767 AND ¬INTER DO
    BEGIN
      R3 := R4 AND #FF SHLL 1;  R4 := FAILSTATE(R3);
      R3 := R4 AND #FF SHLL 1; S := R4;
      R5 := SUCCSTATE(R3) AND #FF SHLL 1; R3 := PRODSTART(R5);
      R5 := R5 - R5; IC(R5,PRODARRAY(R3)); NTSYM2 := R5;
      INTERSECT;
    END;
    R3 := R4 SHRL 8;
    IF R3 = LOOKAHEADORD OR R3 = LOOKAHEADEMPTY THEN
    BEGIN
      NOTSLR1;
    END ELSE
    BEGIN
      R3 := R4 AND #FF SHLL 1; R5 := TREADSTART(R3);
      R6 := R6 - R6; IC(R6,TSYMLIST(R5)); R7 := R7 - R7;
      R3 := S AND #FF;
      IC(R7,TRDNUM(R3)); R7 := R7 + R5;
      R12 := BASEREG008; R3 := R5 SHLL 1; R4 := TSTATELIST(R3);
      R12 := BASE005; SYM := R6; ISVALIDLA;
      IF R4 = #6FF OR ¬ISVALID THEN SET(ERRORST)
      ELSE RESET(ERRORST);
      WHILE R5 < R7 AND ERRORST DO
      BEGIN
        R5 := R5 + 1; R6 := R6 - R6; IC(R6,TSYMLIST(R5));
        R12 := BASEREG008; R3 := R5 SHLL 1; R4 := TSTATELIST(R3);
        R12 := BASE005; SYM := R6; ISVALIDLA;
        IF R4 = #6FF OR ¬ISVALID THEN SET(ERRORST)
        ELSE RESET(ERRORST);
      END;
      IF R5 < R7 THEN NOTSLR1;
    END;
  END;
  MVC(34,WBUF,"LOOK-AHEAD SETS HAVE BEEN COMPUTED.");
  WRITE; MVC(34,WBUF,BLANK);
  IF ISLR1 THEN
  BEGIN
    MVC(20,WBUF,"THE GRAMMAR IS LR(1).");
    WRITE; MVC(20,WBUF,BLANK);
  END;
  WRITE; WRITE;
  R1 := ERRCOUNT;
  IF LASETSLIST OR ¬ISLR1 OR R1 ¬= 0 THEN PRINTSLR1LASETS;
  R14 := SAVE14; R12 := BASE010;
END;  COMMENT  END OF LOOKAHEADAND DPDA;
```

```
GLOBAL PROCEDURE COMPUTEDPDA(R14);
  BEGIN INTEGER SAVE14;

    ARRAY 255 SHORT INTEGER REDSUCCFORNT SYN NEWV(500);
    ARRAY 255 SHORT INTEGER RESUMELIST SYN NEWV(0);
    ARRAY 255 BYTE FREGLIST SYN BSSIZE(0);
    ARRAY 255 SHORT INTEGER LBLIST SYN BASISSTACK(0);

    INTEGER LISTPTR, LBST, RESSTATE, TEMP, LBPTR;
    BYTE NOTFOUND, NEWLBSTATE;
    SHORT INTEGER MN, MNPTR, MX, MXPTR;


    PROCEDURE ENTER(R14);
      BEGIN INTEGER SAVE14; ARRAY 3 INTEGER SAVEREGS;
        SAVE14 := R14; STM(R1,R3,SAVEREGS); R1 := 0;
        IF R1 < LISTPTR THEN SET(NOTFOUND)
        ELSE RESET(NOTFOUND);
        WHILE NOTFOUND DO
        BEGIN
          R2 := R1 SHLL 1; R3 := RESUMELIST(R2);
          IF R3 = RESSTATE THEN
          BEGIN
            IC(R2,FREGLIST(R1)); R2 := R2 + 1;
            STC(R2,FREGLIST(R1)); RESET(NOTFOUND);
          END ELSE
          BEGIN
            R1 := R1 + 1;
            IF R1 < LISTPTR THEN SET(NOTFOUND)
            ELSE RESET(NOTFOUND);
          END;
        END;
          R2 := LISTPTR;
          IF R1 = LISTPTR THEN
          BEGIN
            R3 := 1; STC(R3,FREGLIST(R2));
          END;
          R2 := LISTPTR SHLL 1;
        R3 := LBST; STH(R3,LBLIST(R2)); R3 := RESSTATE;
        STH(R3,RESUMELIST(R2)); R2 := LISTPTR + 1; LISTPTR := R2;
        R14 := SAVE14; LM(R1,R3,SAVEREGS);
      END; COMMENT END OF ENTER;

    PROCEDURE INCLBPTR(R14);
      BEGIN INTEGER SAVE14;
        SAVE14 := R14; R14 := LBPTR + 1;
        IF R14 < 256 THEN LBPTR := R14
```

```
    ELSE BEGIN
      MVC(29,WBUF(15),"TOO MANY LOOK BACK TRANSITION");
      ERROR;
    END;
    R14 := SAVE14;
  END;  COMMENT  END OF INCLBPTR;

PROCEDURE INCNUMLBSTATES(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14; R14 := NUMLBSTATES + 1;
    IF R14 < 128 THEN NUMLBSTATES := R14
    ELSE BEGIN
      MVC(24,WBUF(15),"TOO MANY LOOK BACK STATES");
      ERROR;
    END;
    R14 := SAVE14;
  END;  COMMENT  END OF INCNUMLBSTATES;

PROCEDURE ENCODELB(R14);
  BEGIN INTEGER SAVE14; ARRAY 2 INTEGER SAVEREGS;
    SAVE14 := R14; STM(R1,R2,SAVEREGS);
    R1 := 255; MN := R1; MNPTR := R1;
    R1 := 0; MX := R1; MXPTR := R1; R7 := LISTPTR - 1;
    FOR R1 := 0 STEP 1 UNTIL R7 DO
    BEGIN
      R2 := R2 - R2; IC(R2,FREGLIST(R1));
      IF R2 < MN AND R2 ¬= 0 THEN
      BEGIN
        MN := R2; MNPTR := R1;
      END;
      IF R2 >= MX THEN
      BEGIN
        MX := R2; MXPTR := R1;
      END;
    END;
    RESET(NEWLBSTATE); R1 := MNPTR;
    IF R1 ¬= MXPTR THEN
    BEGIN
      SET(NEWLBSTATE); R1 := NUMLBSTATES; R2 := LBPTR;
      STC(R2,LBSTART(R1)); R2 := 0; STC(R2,LBNUM(R1));
    END;
    R1 := MNPTR;
    WHILE R1 ¬= MXPTR DO
    BEGIN
      R2 := 0; STC(R2,FREGLIST(R1)); R2 := NUMLBSTATES;
      R3 := R3 - R3; IC(R3,LBNUM(R2)); R3 := R3 + MN;
      STC(R3,LBNUM(R2));  R2 := LISTPTR - 1 SHLL 1;
      FOR R3 := 0 STEP 2 UNTIL R2 DO
```

```
      BEGIN
        R4 := MNPTR SHLL 1; R5 := RESUMELIST(R3);
        IF R5 = RESUMELIST(R4) THEN
        BEGIN
          R4 := LBPTR SHLL 1; R5 := RESUMELIST(R3);
          RESUMESTATE(R4) := R5; R5 := LBLIST(R3);
          LBSTATE(R4) := R5; INCLBPTR;
        END;
      END;
      R2 := 255; MN := R2; R3 := LISTPTR - 1;
      FOR R4 := 0 STEP 1 UNTIL R3 DO
      BEGIN
        R2 := R2 - R2; IC(R2,FREGLIST(R4));
        IF R2 < MN AND R2 ¬= 0 THEN
        BEGIN
          MN := R2; MNPTR := R4; R1 := R4;
        END;
      END;
    END;
    IF NEWLBSTATE THEN
    BEGIN
      R1 := MXPTR SHLL 1; R2 := LBPTR SHLL 1;
      R3 := RESUMELIST(RI); RESUMESTATE(R2) := R3; INCLBPTR;
      R1 := LOOKBACK SHLL 8 OR NUMLBSTATES;
      R2 := SYM SHLL 1; REDSUCCFORNT(R2) := R1;
      INCNUMLBSTATE;
    END ELSE
    BEGIN
      R1 := RESUMELIST(0); R2 := SYM SHLL 1;
      REDSUCCFORNT(R2) := R1;
    END;
    R14 := SAVE14; LM(R1,R2,SAVEREGS);
  END;  COMMENT  END OF ENCODELB;

PROCEDURE PRINTDPDA(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14;
    MVC(39,WBUF,"THE DPDA FOR THE GRAMMAR CONSISTS OF THE");
    MVC(36,WBUF(41),"TERMINAL TRANSITIONS OF THE CFSM PLUS");
    RO := @WBUF; WRITE; MVC(80,WBUF,BLANK);
    MVC(33,WBUF,"THE FOLLOWING REDUCE AND LOOK-BACK");
    MVC(10,WBUF(35),"TRANSITIONS");
    WRITE; MVC(50,WBUF,BLANK); WRITE; WRITE;
    FOR R1 := 1 STEP 1 UNTIL NUMPRODS DO
    BEGIN
      R2 := R1 SHLL 1; R3 := REDUCESUCC(R2);
      S:= R3; CONVSTATE; MVC(11,WBUF,"STATE REDUCE");
      R2 := 13; CVD(R1,CONWORK);
```

```
      IF R1 < 10 THEN UNPK(0,7,WBUF(13),CONWORK)
      ELSE BEGIN
        UNPK(1,7,WBUF(13),CONWORK); R2 := 14;
      END;
    R3 := @WBUF(R2); SETZONE(B3); R3 := R3 + 1;
      MVC(0,B3,":"); MVC(2,WBUF(21),"POP"); R3 := R3 - R3;
      R12 := BASE005;
      IC(R3,RTPTSIZE(R1)); R3 := R3 - 1; CVD(R3,CONWORK);
      R12 := BASEREG008;
      UNPK(1,7,WBUF(25),CONWORK); SETZONE(WBUF(26));
      MVC(1,WBUF(38),ARROW); MVC(24,WBUF(42),TBUF);
      WRITE; MVC(70,WBUF,BLANK); WRITE;
    END;
    WRITE;
    R2 := NUMLBSTATES - 1;
    FOR R1 := 0 STEP 1 UNTIL R2 DO
    BEGIN
      MVC(14,WBUF,"STATE LOOK BACK"); CVD(R1,CONWORK);
      UNPK(1,7,WBUF(16),CCNWORK);
      SETZONE(WBUF(17)); MVC(0,WBUF(18),":"); WRITE;
      MVC(18,WBUF,BLANK); R3 := R3 - R3; IC(R3,LBSTART(R1));
      R4 := R4 - R4; IC(R4,LBNUM(R1));
      R7 := R3 + R4 - 1;
      FOR R3 := R3 STEP 1 UNTIL R7 DO
      BEGIN
        R4 := R3 SHLL 1; R5 := LBSTATE(R4); S := R5; CONVSTATE;
        MVC(24,WBUF(16),TBUF); MVC(1,WBUF(38),ARROW);
        R5 := RESUMESTATE(R4); S := R5; CONVSTATE;
        MVC(24,WBUF(42),TBUF); WRITE; MVC(70,WBUF,BLANK);
      END;
      R4 := R3 SHLL 1; R5 := RESUMESTATE(R4);
      S := R5; CONVSTATE; MVC(6,WBUF(16),"DEFAULT");
      MVC(1,WBUF(38),ARROW); MVC(24,WBUF(42),TBUF);
      WRITE; MVC(70,WBUF,BLANK); WRITE; WRITE;
    END;
    R14 := SAVE14;
  END;  COMMENT  END OF PRINT DPDA;


SAVE14 := R14; R8 := BASE010; R12 := BASEREG008;
R1 := 0; NUMLBSTATES := R1; LBPTR := R1;
R1 := NUMREADSTATES + 1; LISTPTR := R1;
R2 := NUMTERMINALS + NUMNTS; TEMP := R2;
FOR R1 := NUMTERMINALS + 1 STEP 1 UNTIL TEMP DO
BEGIN
  SYM := R1; R2 := 0; R4 := LISTPTR - 1;
  FOR R3 := 0 STEP 1 UNTIL R4 DO STC(R2,FREGLIST(R3));
  LISTPTR := R2; R7 := NUMREADSTATES SHLL 1;
```

```
      FOR R3 := 0 STEP 2 UNTIL R7 DO
      BEGIN
        R2 := R2 - R2; R4 := R3 SHRL 1; IC(R2,NTRDNUM(R4));
        R2 := R2 + NTREADSTART(R3) - 1;
        FOR R4 := NTREADSTART(R3) STEP 1 UNTIL R2 DO
        BEGIN
          R5 := R5 - R5; IC(R5,NTSYMLIST(R4));
          IF R5 = SYM THEN
          BEGIN
            R4 := R4 SHLL 1;
            R5 := LINEARTOARRAY(R3); R6 := NTSTATELIST(R4);
            R4 := R4 SHRL 1;
            LBST := R5; RESSTATE := R6; ENTER;
            R5 := R5 - R5; R6 := R3 SHRL 1; IC(R5,NTRDNUM(R6));
            R4 := R4 + R5;
          END;
        END;
      END;
      ENCODELB;
    END;
    R2 := NUMREDUCESTATES - 1 SHLL 1;
    FOR R1 := 0 STEP 2 UNTIL R2 DO
    BEGIN
      R3 := PRODSTART(R1); R4 := R4 - R4; R12 := BASE005;
      IC(R4,PRODARRAY(R3)); R12 := BASEREG008; R4 := R4 SHLL 1;
      R5 := REDSUCCFORNT(R4); REDUCESUCC(R1) := R5;
    END;
    R1 := R1 + 2; R2 := EXIT SHLL 8; REDUCESUCC(R1) := R2;
    MVC(26,WBUF,"THE DPDA HAS BEEN COMPUTED,"); WRITE;
    MVC(26,WBUF,BLANK); WRITE;
    IF DPDALIST THEN PRINTDPDA;
    R14 := SAVE14;
  END;  COMMENT  END OF COMPUTEDPDA;

GLOBAL PROCEDURE PUNCHDPDA(R14);
  BEGIN
    INTEGER SAVE14, NUMLINE, HEXLENGTH, DECLENGTH, NUMBITS, CP;
    INTEGER ADDRESS, ARRAYLENGTH, TEMP;
    ARRAY 16 BYTE HEXCODE = ("0123456789ABCDEF");
    ARRAY 10 BYTE HEXLINE = "#";
    BYTE ARRAYEND, LL;
    ARRAY 18 INTEGER SEGTABLE;
    INTEGER BYTECNT = 0;
    INTEGER SEGPT = 0;


    PROCEDURE PUNCHCOMMENT(R14);
      BEGIN INTEGER SAVE14;
```

```
              SAVE14 := R14; CVD(R1,CONWORK);
              R3 := @WBUF(R2); UNPK(3,7,B3,CONWORK); R3 := R3 + 3;
              SETZONE(B3);   WRITE; PUNCH; MVC(80,WBUF,BLANK);
              WRITE; PUNCH; R14 := SAVE14;
            END;  COMMENT  END OF PUNCHCOMMENT;

          PROCEDURE PUNCHDECLARE(R14);
            BEGIN INTEGER SAVE14; SAVE14 := R14;
              R1 := ARRAYLENGTH + 1;
              MVC(4,WBUF(2),"ARRAY"); CVD(R1,CONWORK);
              UNPK(3,7,WBUF(8),CONWORK); SETZONE(WBUF(11));
              R14 := NUMBITS;
              IF LL THEN
              BEGIN
                MVC(6,WBUF(13),"INTEGER"); R1 := 21; GOTO JUMP;
              END;
              IF R14 = 32 THEN
              BEGIN
                MVC(3,WBUF(13),"REAL"); R1 := 18; GOTO JUMP;
              END;
              IF R14 = 8 THEN
              BEGIN
                MVC(3,WBUF(13),"BYTE"); R1 := 18;
              END ELSE
              BEGIN
                MVC(12,WBUF(13),"SHORT INTEGER"); R1 := 27;
              END;
    JUMP:     FOR R2 := 0 STEP 1 UNTIL DECLENGTH DO
              BEGIN
                IC(R3,TBUF(R2)); STC(R3,WBUF(R1)); R1 := R1 + 1;
              END;
              R1 := R1 + 1; R2 := @WBUF(R1);
              MVC(2,B2,"= ("); R1 := R1 + 3; CP := R1;
              R14 := SAVE14;
            END;, COMMENT  END OF PUNCHDECLARE;

          PROCEDURE BUILDHEXLINE(R14);
            BEGIN INTEGER SAVE14; ARRAY 5 INTEGER SAVEREGS;
              SAVE14 := R14; STM(R1,R5,SAVEREGS);
              R14 := NUMLINE;
              R3 := NUMBITS - 4;
              R2 := NUMBITS SHRL 2;
              FOR R1 := 0 STEP 4 UNTIL R3 DO
              BEGIN
                R5 := NUMLINE SHRL R1 AND #F;
                IC(R4,HEXCODE(R5)); STC(R4,HEXLINE(R2)); R2 := R2 - 1;
              END;
              R1 := NUMBITS;
```

```
    IF R1 = 16 THEN MVC(0,HEXLINE(5),"S");
    IF R1 = 8 THEN MVC(0,HEXLINE(3),"X");
    R14 := SAVE14; LM(R1,R5,SAVEREGS);
  END;  COMMENT  END OF BUILDHEXLINE;

PROCEDURE PUNCHCARD(R14);
  BEGIN INTEGER SAVE14; SAVE14 := R14;
    R14 := CP + HEXLENGTH;
    IF R14 >= 70 THEN
    BEGIN
      R0 := @WBUF; WRITE; PUNCH; MVC(72,WBUF,BLANK); R14 := 10;
    END ELSE R14 := CP;
    FOR R7 := 0 STEP 1 UNTIL HEXLENGTH DO
    BEGIN
      IC(R6,HEXLINE(R7)); STC(R6,WBUF(R14)); R14 := R14 + 1;
    END;
    IF ARRAYEND THEN
    BEGIN
      IC(R7,")"); STC(R7,WBUF(R14)); R14 := R14 + 1;
      IC(R7,";"); STC(R7,WBUF(R14));  WRITE; PUNCH;
      MVC(72,WBUF,BLANK); R14 := 10;
    END ELSE
    BEGIN
      IC(R7,","); STC(R7,WBUF(R14));R14 := R14 + 2;
    END;
    CP := R14; R14 := SAVE14;
  END;  COMMENT  OF PUNCHCARD;

PROCEDURE PUNCHARRAY(R14);
  BEGIN INTEGER SAVE14;
    SAVE14 := R14;
    PUNCHDECLARE;
    R1 := ADDRESS; RESET(ARRAYEND);
    R2 := NUMBITS;
    IF R2 = 32 THEN
    BEGIN
      R2 := 8; HEXLENGTH := R2;
      R2 := ARRAYLENGTH SHLL 2;
      FOR R3 := 0 STEP 4 UNTIL R2 DO
      BEGIN
        LOAD(R4,B1); NUMLINE := R4;
        IF R3 = R2 THEN SET(ARRAYEND) ELSE RESET(ARRAYEND);
        BUILDHEXLINE; PUNCHCARD; R1 := R1 + 4;
      END;
      R2 := ARRAYLENGTH + 1 SHLL 2;
      GOTO ENDARRAY;
    END;
    IF R2 = 8 THEN
```

```
         BEGIN
           R2 := 3; HEXLENGTH := R2;
           FOR R3 := 0 STEP 1 UNTIL ARRAYLENGTH DO
           BEGIN
             R2 := R2 - R2; IC(R2,B1); NUMLINE := R2;
             IF R3 = ARRAYLENGTH THEN SET(ARRAYEND);
             BUILDHEXLINE; PUNCHCARD; R1 := R1 + 1;
           END;
           R2 := ARRAYLENGTH + 1;
         END ELSE
         BEGIN
           R2 := 5; HEXLENGTH := R2;
           R2 := ARRAYLENGTH SHLL 1;
           FOR R3 := 0 STEP 2 UNTIL R2 DO
           BEGIN
             LH(R4,B1); NUMLINE:= R4;
             IF R3 = R2 THEN SET(ARRAYEND);
             BUILDHEXLINE; PUNCHCARD; R1 := R1 + 2;
           END;
           R2 := ARRAYLENGTH + 1 SHLL 1;
         END;
ENDARRAY:
         R1 := NUMBITS; R3 := BYTECNT;
         IF R1 > 8 THEN
         BEGIN
           IF R1 = 16 THEN
           BEGIN
             R4 := R3 AND #1;
             IF R4 > 0 THEN R4 := R3 + 1
             ELSE R4 := R3;
           END ELSE
           BEGIN
             R4 := R3 AND #3;
             IF R4 > 0 THEN R4 := R3 AND #FFFFFFFC + 4
             ELSE R4 := R3;
           END;
         END ELSE R4 := R3;
         R3 := SEGPT SHLL 2; SEGTABLE(R3) := R4;
         R3 := R3 SHRL 2 + 1; SEGPT := R3;
         R4 := R4 + R2; BYTECNT := R4;
         WRITE; PUNCH;
         R14 := SAVE14;
       END;  COMMENT  END OF PUNCHARRAY;

     PROCEDURE PUNCHLASETS(R14);
       BEGIN INTEGER SAVE14;
         SAVE14 := R14; R12 := BASE005; R8 := BASE011;
         RESET(ARRAYEND);
```

```
R4 := BYTECNT;
R3 := SEGPT SHLL 2; SEGTABLE(R3) := R4;
BYTECNT := R4; R3 := R3 SHRL 2 + 1; SEGPT := R3;
R1 := 8;  NUMBITS := R1; MVC(6,TBUF,"LATABLE");
R1 := 6; DECLENGTH := R1;
R1 := NUMTERMINALS + 1 SHRL 3;
R2 := NUMTERMINALS + 1 AND #7;
IF R2 > 0 THEN R1 := R1 + 1;
R1 := R1 * NUMLASTATES - 1; ARRAYLENGTH := R1;
PUNCHDECLARE;
R0 := @WBUF; WRITE; PUNCH; MVC(28,WBUF,BLANK);
R14 := NUMLASTATES - 1; TEMP := R14;
FOR R1 := 0 STEP 1 UNTIL TEMP DO
BEGIN
  R2 := R1 SHLL 1; R3 := SUCCSTATE(R2) AND #FF SHLL 1;
  R2 := PRODSTART(R3); R3 := R3 - R3; IC(R3,PRODARRAY(R2));
  NTSYM := R3; R7 := 0;
  R2 := NTSYM; VPT := R2; FIND; R3 := 19;
  FOR R2 := 0 STEP 1 UNTIL CLENGTH DO
  BEGIN
    IC(R4,CBCD(R2)); STC(R4,WBUF(R3)); R3 := R3 + 1;
  END;
  MVC(6,WBUF(10),"COMMENT"); IC(R4,";"); STC(R4,WBUF(R3));
  R0 := @WBUF; WRITE; PUNCH; MVC(80,WBUF,BLANK);
  R7 := 8; NUMBITS := R7; R6 := 3;
  HEXLENGTH := R6; R6 := 10; CP := R6; R6 := 1;
  R7 := 0;
  FOR R2 := 1 STEP 1 UNTIL NUMTERMINALS DO
  BEGIN
    SYM := R2; ISVALIDLA; R7 := R7 SHLL 1;
    IF ISVALID THEN R7 := R7 OR #1;
    R6 := R6 + 1;
    IF R2 = NUMTERMINALS AND R1 = TEMP THEN SET(ARRAYEND);
    IF R6= 8 THEN
    BEGIN
      NUMLINE := R7; BUILDHEXLINE; PUNCHCARD;
      R6 := BYTECNT + 1; BYTECNT := R6; R6 := 0; R7 := 0;
    END;
  END;
  IF R6 > 0 THEN
  BEGIN
    R2 := NUMTERMINALS AND #7; R3 := 7 - R2;
    R7 := R7 SHLL R3; R2 := BYTECNT + 1; BYTECNT := R2;
    NUMLINE := R7; BUILDHEXLINE; PUNCHCARD;
  END;
  WRITE; PUNCH; MVC(80,WBUF,BLANK); WRITE; PUNCH;
END;
WRITE; PUNCH;
```

```
     R14 := SAVE14;
   END;  COMMENT  END OF PUNCH LASETS;


SAVE14 := R14; RESET(LL);
MVC(21,WBUF(2),"INTEGER NUMTERMINALS ="); R1 := NUMTERMINALS;
CVD(R1,CONWORK); UNPK(3,7,WBUF(25),CONWORK); SETZONE(WBUF(28));
MVC(0,WBUF(29),";"); R0 := @WBUF; WRITE; PUNCH;
MVC(29,WBUF,BLANK); MVC(15,WBUF(2),"INTEGER NUMNTS =");
R1 := NUMNTS; CVD(R1,CONWORK); UNPK(3,7,WBUF(19),CONWORK);
SETZONE(WBUF(22)); MVC(0,WBUF(23),";"); WRITE; PUNCH;
MVC(23,WBUF,BLANK); MVC(16,WBUF(2),"INTEGER NUMSYMS =");
R1 := NUMSYMS + 1; CVD(R1,CONWORK); UNPK(3,7,WBUF(20),CONWORK);
SETZONE(WBUF(23));
MVC(0,WBUF(24),";"); WRITE; PUNCH; MVC(24,WBUF,BLANK);
WRITE; WRITE; PUNCH; PUNCH;
MVC(30,WBUF(2),"GLOBAL DATA SEGTABLE BASE R12; ");
WRITE; PUNCH; MVC(30,WBUF(2),BLANK);
WRITE; PUNCH;
R12 := BASE005;
R11 := BASE004;
MVC(8,V(0),"ERRORSYM ");
R1 := VLENGTH; ARRAYLENGTH := R1; R1 := @V;
ADDRESS := R1; R1 := 8; NUMBITS := R1; MVC(6,TBUF,"VSTRING");
R1 := 6; DECLENGTH := R1; PUNCHARRAY;
R1 := NUMTERMINALS + NUMNTS; ARRAYLENGTH := R1;
R1 := @LOCLENGTH; ADDRESS := R1; R1 := 32; NUMBITS := R1;
SET(LL);
MVC(8,TBUF,"LOCLENGTH"); R1 := 8; DECLENGTH := R1; PUNCHARRAY;
RESET(LL); R11 := BASE006;
MVC(20,WBUF(2),"COMMENT  THE DPDA HAS");
MVC(11,WBUF(29),"READ STATES;");
R1 := NUMREADSTATES + 1; R2 := 24;
PUNCHCOMMENT;
R1 := @TREADSTART; ADDRESS := R1; R1 := 16; NUMBITS := R1;
R1 := NUMREADSTATES; ARRAYLENGTH := R1; MVC(8,TBUF,"READSTART");
R5 := 8; DECLENGTH := R5; PUNCHARRAY;
R1 := @TRDNUM; ADDRESS := R1; R1 := 8; NUMBITS := R1;
MVC(4,TBUF,"RDNUM"); R1 := 4; DECLENGTH := R1; PUNCHARRAY;
R1 := @TSYMLIST; ADDRESS := R1; R1 := NUMREADSTATES;
R2 := R2 - R2; IC(R2,TRDNUM(R1)); R1 := R1 SHLL 1;
R2 := R2 + TREADSTART(R1); ARRAYLENGTH := R2;
MVC(6,TBUF,"SYMLIST"); R1 := 6; DECLENGTH := R1; PUNCHARRAY;
R12 := BASEREG008; R1 := @TSTATELIST; ADDRESS:= R1;
R1 := 16; NUMBITS := R1; MVC(8,TBUF,"STATELIST");
R1 := 8; DECLENGTH := R1; PUNCHARRAY;

MVC(20,WBUF(2),"COMMENT  THE DPDA HAS");
```

```
MVC(13,WBUF(29),"REDUCE STATES;"); R1 := NUMPRODS; R2 := 24;
PUNCHCOMMENT;
R12 := BASE005;
FOR R1 := 1 STEP 1 UNTIL NUMPRODS DO
BEGIN
  R2 := R2 - R2; IC(R2,RTPTSIZE(R1));
  IF R2 > 0 THEN
  BEGIN
    R2 := R2 - 1; STC(R2,RTPTSIZE(R1));
  END;
END;
R1 := @RTPTSIZE; ADDRESS := R1; R1 := NUMPRODS;
ARRAYLENGTH := R1; R1 := 8; NUMBITS := R1;
MVC(7,TBUF,"NUMTOPOP"); R1 := 7; DECLENGTH := R1; PUNCHARRAY;
R1 := @REDUCESUCC; ADDRESS := R1; R1 := 16; NUMBITS := R1;
MVC(9,TBUF,"REDUCESUCC"); R1 := 9; DECLENGTH := R1; PUNCHARRAY;

MVC(20,WBUF(2),"COMMENT  THE DPDA HAS");
MVC(17,WBUF(29),"LOOK AHEAD STATES;");
R1 := NUMLASTATES; R2 := 24; PUNCHCOMMENT;

R1 := @LASYMNUM; ADDRESS := R1; R1 := 8; NUMBITS := R1;
R1 := NUMLASTATES - 1; ARRAYLENGTH := R1; R1 := 7;
DECLENGTH := R1; MVC(7,TBUF,"LASYMNUM"); PUNCHARRAY;
R1 := @SUCCSTATE; ADDRESS := R1; R1 := 16; NUMBITS := R1;
MVC(8,TBUF,"SUCCSTATE"); R1 := 8; DECLENGTH := R1; PUNCHARRAY;
R1 := @FAILSTATE; ADDRESS := R1;
MVC(8,TBUF,"FAILSTATE"); PUNCHARRAY;

PUNCHLASETS;

MVC(20,WBUF(2),"COMMENT  THE DPDA HAS");
MVC(15,WBUF(29),"LOOKBACK STATES;");
R1 := NUMLBSTATES; R2 := 24; PUNCHCOMMENT;

R1 :=' NUMLBSTATES - 1;
IF R1 > 0 THEN
BEGIN
  ARRAYLENGTH := R1; R1 := @LBSTART; ADDRESS := R1; R1 := 8;
  NUMBITS := R1; MVC(6,TBUF,"LBSTART"); R1 := 6;
  DECLENGTH := R1; PUNCHARRAY;
  R1 := @LBNUM; ADDRESS := R1; MVC(4,TBUF,"LBNUM");
  R1 := 4; DECLENGTH := R1;
  PUNCHARRAY;
END ELSE
BEGIN
  R4 := 0; R3 := SEGPT SHLL 2; SEGTABLE(R3) := R4;
  R3 := R3 + 4; SEGTABLE(R3) := R4;
```

```
      R3 := R3 SHRL 2 + 1; SEGPT := R3;
    END;
    R1 := NUMLBSTATES - 1; R2 := R2 - R2; IC(R2,LBNUM(R1));
    R3 := R3 - R3; IC(R3,LBSTART(R1)); R2 := R2 + R3;
    ARRAYLENGTH := R2;
    R12 := BASEREG008; R1 := @LBSTATE; ADDRESS := R1; R1 := 16;
    NUMBITS := R1; MVC(6,TBUF,"LBSTATE"); R1 := 6;
    DECLENGTH := R1; PUNCHARRAY;
    R1 := @RESUMESTATE; ADDRESS:= R1;
    MVC(10,TBUF,"RESUMESTATE"); R1 := 10; DECLENGTH := R1;
    PUNCHARRAY;
    MVC(41,WBUF(2),"COMMENT  THE SYMBOLS ACCESSING THE STATES;");
    WRITE; PUNCH; MVC(43,WBUF,BLANK); WRITE; PUNCH;
    R8 := BASE010; R1 := @SYMBEFORE; ADDRESS := R1;
    R1 := NUMREADSTATES; ARRAYLENGTH := R1; R1 := 8; NUMBITS := R1;
    MVC(12,TBUF,"SYMBEFOREREAD"); R1 := 12; DECLENGTH := R1;
    PUNCHARRAY;
    R1 := ADDRESS + #300; ADDRESS := R1; R1 := NUMLASTATES - 1;
    ARRAYLENGTH := R1; MVC(10,TBUF,"SYMBEFORELA"); R1 := 10;
    DECLENGTH := R1; PUNCHARRAY;
    MVC(10,WBUF(2),"CLOSE BASE;"); WRITE; PUNCH;
    MVC(10,WBUF(2),BLANK); WRITE; PUNCH;
    MVC(28,WBUF(2),"ARRAY 18 INTEGER SEGTABLE = (");
    WRITE; PUNCH; MVC(28,WBUF(2),BLANK); R2 := 17 SHLL 2;
    R1 := 32; NUMBITS := R1;
    FOR R1 := 0 STEP 4 UNTIL R2 DO
    BEGIN
      R3 := SEGTABLE(R1); NUMLINE := R3;
      BUILDHEXLINE; MVC(8,WBUF(10),HEXLINE);
      IF R1 = R2 THEN MVC(1,WBUF(19),");")
      ELSE MVC(0,WBUF(19),",");
      WRITE; PUNCH; MVC(10,WBUF(10),BLANK);
    END;
    MVC(42,WBUF(2),"COMMENT  END OF CARDS PUNCHED BY THE SLR(1)");
    MVC(15;WBUF(46),"SYNTAX ANALYZER;"); WRITE; PUNCH;
    MVC(61,WBUF,BLANK); WRITE; PUNCH;
    R14 := SAVE14;
  END;  COMMENT  END OF PUNCHDPDA;

COMMENT  MAIN PROGRAM;

CCMMENT  THE FOLLOWING 5 LINES FIND AND SET BASE ADDRESS FOR DATA
SEGMENTS.  THEY SHOULD BE CHANGED ONLY WHEN THE DATA SEGMENTS
HAVE BEEN CHANGED;


R1 := @TSTATELIST; BASEREG008 := R1;
R1 := R11 - #D00; BASE005 := R1;
```

```
R1 := R1 - #1000; BASE004 := R1; BASE006 := R11;
R12 := BASE005;
R1 := R8 - #E00; BASE010 := R1; BASE011 := R8; R8 := BASE010;
STM(R7,R11,SAVEBASEREGS);
SET(MOREGRAMMERS);
WHILE MOREGRAMMERS DO
BEGIN
  R1 := 0; ERRCOUNT := R1;
  RESET(INPUTLIST);  COMMENT  DO NOT LIST GRAMMER CARDS;
  SET(GRAMMERLIST);  COMMENT  LIST REFORMATTED GRAMMER;
  RESET(CONFIGLIST);  COMMENT  DO NOT LIST CONFIGURATION SETS;
  SET(FSMLIST);  COMMENT  LIST CHARACTERISTIC FSM;
  SET(LASETSLIST);  COMMENT  LIST LOOK-AHEAD SETS;
  SET(DPDALIST);  COMMENT  LIST THE DPDA;
  RESET(PUNCHDECK);  COMMENT  DO NOT PUNCH THE DPDA;

  MVC(24,WBUF,"START READING THE GRAMMAR");
  RO := @WBUF; WRITE; MVC(24,WBUF,BLANK);
  READG;
  MVC(24,WBUF,"THE GRAMMAR HAS BEEN READ");
  RO := @WBUF; WRITE; MVC(24,WBUF,BLANK);
  R1 := ERRCOUNT;
  IF R1¬=0 THEN
  BEGIN
    IF R1 = 1 THEN  MVC(19,WBUF,"THERE WAS ONE ERROR,")
    ELSE
    BEGIN
      MVC(19,WBUF,"THERE WERE    ERRORS,");
      CVD(R1,CONWORK); UNPK(1,7,WBUF(11),CONWORK);
      SETZONE(WBUF(12));
    END;
    WRITE;
    MVC(42,WBUF,"THEREFORE THE GRAMMAR WILL NOT BE ANALYZED.");
    WRITE;
    MVC(41,WBUF,"THE GRAMMAR IS LISTED ABOVE FOR DEBUGGING");
    MVC(8,WBUF(43),"PURPOSES.");
    WRITE; MVC(50,WBUF,BLANK); WRITE; WRITE;
  END;
  R1 := ERRCOUNT;
  IF R1 = 0 THEN
  BEGIN
    MVC(31,WBUF,"START CONSTRUCTING THE GRAMMAR'S");
    MVC(18,WBUF(35),"CHARACTERISTIC FSM.");
    RO := @WBUF; WRITE; MVC(53,WBUF,BLANK);
    SET(ISLRO); SET(ISLR1);
    LM(R7,R11,SAVEBASEREGS);
    COMPUTECFSM;
    MVC(42,WBUF,"THE CFSM FOR THE GRAMMARS HAS BEEN COMPUTED.");
```

```
   RO := @WBUF; WRITE; MVC(42,WBUF,BLANK);
   PAGE;
   R1 := ERRCOUNT;
   IF FSMLIST OR R1 ¬= O THEN PRINTCFSM;
  END;
  R1 := ERRCOUNT;
  IF R1 ¬= O THEN
  BEGIN
   MVC(27,WBUF,"SINCE THE CFSM IS IN ERROR,");
   MVC(27,WBUF(28),"ANALYSIS WILL BE TERMINATED.");
   WRITE;  MVC(54,WBUF,BLANK);
  END ELSE
  BEGIN
   IF ISLRO THEN
   BEGIN
    MVC(32,WBUF,"SURPRISE    THE GRAMMAR IS LR(O).");
    WRITE; MVC(32,WBUF,BLANK);
   END ELSE
   BEGIN
    MVC(24,WBUF,"THE GRAMMAR IS NOT LR(O).");
    MVC(24,WBUF(27),"LOOK-AHEAD MUST BE ADDED.");
    WRITE; MVC(60,WBUF,BLANK); WRITE; WRITE;
    MVC(31,WBUF,"START COMPUTING LOOK-AHEAD SETS.");
    WRITE; MVC(31,WBUF,BLANK);
    LOOKAHEADANDDPDA;
   END;
  END;
  R1 := ERRCOUNT;
  IF R1 ¬= O THEN
  BEGIN
   MVC(38,WBUF,"SINCE THE LOOK-AHEAD SETS ARE IN ERROR,");
   MVC(28,WBUF(40),"EXECUTION WILL BE TERMINATED.");
   WRITE; MVC(58,WBUF,BLANK);
  END ELSE
  BEGIN
   MVC(24,WBUF,"START COMPUTING THE DPDA.");
   WRITE; MVC(24,WBUF,BLANK);
   COMPUTEDPDA;
  END;
  PAGE;
  IF ¬PUNCHDECK THEN
  BEGIN
   MVC(22,WBUF,"NO DPDA DECK REQUESTED.");
   WRITE; MVC(22,WBUF,BLANK);
  END ELSE PUNCHDPCA;
  WRITE; WRITE; MVC(8,WBUF,"ALL DONE."); WRITE;
 END;
BAILCUT:
```

```
END.                                                                    2198
```

APPENDIX B

SAMPLE GRAMMAR FOR THE SYNTAX ANALYZER

```
@P
@C
@I
<PROGRAM> <STATEMENT LIST> END
<STATEMENT LIST> <STATEMENT>
          <STATEMENT LIST> <STATEMENT>
<STATEMENT> <ASSIGNMENT> ;
<ASSIGNMENT> <VARIABLE> = <EXPR>
<EXPR> <ARITH EXPR>
<ARITH EXPR> <TERM>
          <ARITH EXPR> + <TERM>
          <ARITH EXPR> - <TERM>
<TERM> <PRIMARY>
          <TERM> * <PRIMARY>
          <TERM> / <PRIMARY>
<PRIMARY> <VARIABLE>
          <NUMBER>
<VARIABLE> <IDENTIFIER>
```

APPENDIX C

SYNTAX ANALYZER OUTPUT

```
START READING THE GRAMMAR
                    <PROGRAM> <STATEMENT LIST> END
                    <STATEMENT LIST> <STATEMENT>
                        <STATEMENT LIST> <STATEMENT>
                    <STATEMENT> <ASSIGNMENT> ;
                    <ASSIGNMENT> <VARIABLE> = <EXPR>
                    <EXPR> <ARITH EXPR>
                    <ARITH EXPR> <TERM>
                        <ARITH EXPR> + <TERM>
                        <ARITH EXPR> - <TERM>
                    <TERM> <PRIMARY>
                        <TERM> * <PRIMARY>
                        <TERM> / <PRIMARY>
                    <PRIMARY> <VARIABLE>
                        <NUMBER>
                    <VARIABLE> <IDENTIFIER>

                         T H E   V O C A B U L A R Y

           T E R M I N A L   S Y M B O L S       N O N T E R M I N A L S

       0001  _|_                                 <PROGRAM>
       0002  END                                 <STATEMENT LIST>
       0003  ;                                   <STATEMENT>
       0004  =                                   <ASSIGNMENT>
       0005  +                                   <VARIABLE>
       0006  -                                   <EXPR>
       CC07  *                                   <ARITH EXPR>
       0008  /                                   <TERM>
       0009  <NUMBER>                            <PRIMARY>
       0010  <IDENTIFIER>

                     THE GOAL SYMBOL IS          <PROGRAM>
```

T H E   P R O D U C T I O N S

```
0001  <PROGRAM>   ::=  <STATEMENT LIST>  END

0002  <STATEMENT LIST>   ::=  <STATEMENT>
0C03                       |  <STATEMENT LIST>  <STATEMENT>

0004  <STATEMENT>   ::=  <ASSIGNMENT>  ;

0005  <ASSIGNMENT>   ::=  <VARIABLE>  =  <EXPR>

00C6  <EXPR>   ::=  <ARITH EXPR>

0007  <ARITH EXPR>   ::=  <TERM>
0008                    |  <ARITH EXPR>  +  <TERM>
0009                    |  <ARITH EXPR>  -  <TERM>

0010  <TERM>   ::=  <PRIMARY>
0011             |  <TERM>  *  <PRIMARY>
0012             |  <TERM>  /  <PRIMARY>

0013  <PRIMARY>   ::=  <VARIABLE>
0014                |  <NUMBER>

0015  <VARIABLE>   ::=  <IDENTIFIER>
```

```
SCME STATISTICS ON THE GRAMMAR:

     NUMBER OF TERMINAL SYMBOLS = 0010
     NUMBER CF NONTERMINAL SYMBOLS = 0009
           TOTAL NUMBER OF SYMBOLS = 0019

     NUMBER CF PRODUCTIONS = 0015
THE GRAMMAR HAS BEEN READ
```

```
START CCNSTRUCTING THE GRAMMAR'S    CHARACTERISTIC FSM.
THE CONFIGURATION SET FOR STATE     0000 IS:
    0001  <SYSTEMGS>  -> . _|_  <PROGRAM>  _|_

THE CONFIGURATION SET FOR STATE     0001 IS:
    0001  <SYSTEMGS>  ->  _|_ . <PROGRAM>  _|_
    0002  <PROGRAM>  -> . <STATEMENT LIST>  END
    0003  <STATEMENT LIST>  -> . <STATEMENT LIST>  <STATEMENT>
    0004  <STATEMENT LIST>  -> . <STATEMENT>
    0005  <STATEMENT>  -> . <ASSIGNMENT>  ;
    0006  <ASSIGNMENT>  -> . <VARIABLE>  =  <EXPR>
    0007  <VARIABLE>  -> . <IDENTIFIER>


THE CCNFIGURATION SET FOR STATE     0002 IS:
    0C01  <SYSTEMGS>  ->  _|_  <PROGRAM> . _|_


THE CCNFIGURATION SET FOR STATE     0003 IS:
    0001  <PROGRAM>  ->  <STATEMENT LIST> . END
    0002  <STATEMENT LIST>  ->  <STATEMENT LIST> . <STATEMENT>
    0003  <STATEMENT>  -> . <ASSIGNMENT>  ;
    0004  <ASSIGNMENT>  -> . <VARIABLE>  =  <EXPR>
    0005  <VARIABLE>  -> . <IDENTIFIER>


THE CONFIGURATION SET FOR STATE     0004 IS:
    0001  <STATEMENT>  ->  <ASSIGNMENT> . ;


THE CONFIGURATION SET FOR STATE     0005 IS:
    0001  <ASSIGNMENT>  ->  <VARIABLE> . =  <EXPR>


THE CONFIGURATION SET FOR STATE     0006 IS:
    0001  <ASSIGNMENT>  ->  <VARIABLE>  = . <EXPR>
    0002  <EXPR>  -> . <ARITH EXPR>
    0003  <ARITH EXPR>  -> . <ARITH EXPR>  +  <TERM>
    0004  <ARITH EXPR>  -> . <ARITH EXPR>  -  <TERM>
    0005  <ARITH EXPR>  -> . <TERM>
    0CC6  <TERM>  -> . <TERM>  *  <PRIMARY>
    0007  <TERM>  -> . <TERM>  /  <PRIMARY>
    0008  <TERM>  -> . <PRIMARY>
    0C09  <PRIMARY>  -> . <VARIABLE>
    0010  <PRIMARY>  -> . <NUMBER>
    0011  <VARIABLE>  -> . <IDENTIFIER>
```

```
THE CCNFIGURATION SET FOR STATE    0007 IS:
    0001  <EXPR>  ->  <ARITH EXPR> .
    0002  <ARITH EXPR>  ->  <ARITH EXPR> . +  <TERM>
    0003  <ARITH EXPR>  ->  <ARITH EXPR> . -  <TERM>

THE CCNFIGURATION SET FOR STATE    0008 IS:
    0001  <ARITH EXPR>  ->  <TERM> .
    0002  <TERM>  ->  <TERM> . *  <PRIMARY>
    0003  <TERM>  ->  <TERM> . /  <PRIMARY>

THE CONFIGURATION SET FOR STATE    0009 IS:
    0001  <ARITH EXPR>  ->  <ARITH EXPR>  + . <TERM>
    0002  <TERM>  -> . <TERM>  *  <PRIMARY>
    0003  <TERM>  -> . <TERM>  /  <PRIMARY>
    0004  <TERM>  -> . <PRIMARY>
    0005  <PRIMARY>  -> . <VARIABLE>
    0006  <PRIMARY>  -> . <NUMBER>
    0007  <VARIABLE>  -> . <IDENTIFIER>

THE CCNFIGURATION SET FOR STATE    0010 IS:
    0001  <ARITH EXPR>  ->  <ARITH EXPR>  - . <TERM>
    0002  <TERM>  -> . <TERM>  *  <PRIMARY>
    0003  <TERM>  -> . <TERM>  /  <PRIMARY>
    0004  <TERM>  -> . <PRIMARY>
    0005  <PRIMARY>  -> . <VARIABLE>
    0CC6  <PRIMARY>  -> . <NUMBER>
    0007  <VARIABLE>  -> . <IDENTIFIER>

THE CONFIGURATION SET FOR STATE    0011 IS:
    0001  <TERM>  ->  <TERM>  * . <PRIMARY>
    0002  <PRIMARY>  -> . <VARIABLE>
    0003  <PRIMARY>  -> . <NUMBER>
    0004  <VARIABLE>  -> . <IDENTIFIER>

THE CCNFIGURATION SET FOR STATE    0012 IS:
    0001  <TERM>  ->  <TERM>  / . <PRIMARY>
    0C02  <PRIMARY>  -> . <VARIABLE>
    0003  <PRIMARY>  -> . <NUMBER>
    0004  <VARIABLE>  -> . <IDENTIFIER>

THE CONFIGURATION SET FOR STATE    0013 IS:
    0001  <ARITH EXPR>  ->  <ARITH EXPR>  +  <TERM> .
```

```
    0002  <TERM>  ->  <TERM> . *  <PRIMARY>
    0003  <TERM>  ->  <TERM> . /  <PRIMARY>

THE CONFIGURATION SET FOR STATE    0014 IS:
    0001  <ARITH EXPR>  ->  <ARITH EXPR>  -  <T ERM> .
    0002  <TERM>  ->  <TERM> . *  <PRIMARY>
    0003  <TERM>  ->  <TERM> . /  <PRIMARY>
```

```
THE CFSM FOR THE GRAMMARS HAS BEEN COMPUTED
THE CFSM FOR THE GRAMMAR IS AS FOLLOWS

STATE READ LINEAR       0000:    ACCESSING SYMBOL IS  <SYSTEMGS>)

                                   _|_  -> READ LINEAR       0001
          -----------------------------------------------

STATE READ LINEAR       0001:    ACCESSING SYMBOL IS  _|_)

                          <IDENTIFIER>  -> REDUCE            0015
          -----------------------------------------------
                             <PROGRAM>  -> READ LINEAR       0002
                      <STATEMENT LIST>  -> READ LINEAR       0003
                           <STATEMENT>  -> REDUCE            0002
                          <ASSIGNMENT>  -> READ LINEAR       0004
                            <VARIABLE>  -> READ LINEAR       0005

STATE READ LINEAR       0002:    ACCESSING SYMBOL IS  <PROGRAM>)

                                   _|_  -> REDUCE            0016
          -----------------------------------------------

STATE READ LINEAR       0003:    ACCESSING SYMBOL IS  <STATEMENT LIST>)

                                   END  -> REDUCE            0001
                          <IDENTIFIER>  -> REDUCE            0015
          -----------------------------------------------
                           <STATEMENT>  -> REDUCE            0003
                          <ASSIGNMENT>  -> READ LINEAR       0004
                            <VARIABLE>  -> READ LINEAR       0005

STATE READ LINEAR       0004:    ACCESSING SYMBOL IS  <ASSIGNMENT>)

                                     ;  -> REDUCE            0004
          -----------------------------------------------

STATE READ LINEAR       0005:    ACCESSING SYMBOL IS  <VARIABLE>)

                                     =  -> READ LINEAR       0006
          -----------------------------------------------
```

```
STATE READ LINEAR        0006:    ACCESSING SYMBOL IS  =)

                         <NUMBER>  -> REDUCE                  0014
                     <IDENTIFIER>  -> REDUCE                  0015
            ----------------------------------------------
                       <VARIABLE>  -> REDUCE                  0013
                           <EXPR>  -> REDUCE                  0005
                     <ARITH EXPR>  -> LOOK AHEAD ORD          0000
                           <TERM>  -> LOOK AHEAD ORD          0001
                        <PRIMARY>  -> REDUCE                  0010

READ STATE   0007 IS INADEQUATE   THE FOLLOWING LOOK-AHEAD IS NECESSARY.

LOOK AHEAC ORD    0000  :        LA SYM NUM = 0001)

LA SET OF                          <EXPR>  -> REDUCE          0006
                                  DEFAULT  -> READ LINEAR     0007 )

STATE READ LINEAR        0007:)   ACCESSING SYMBOL IS  <ARITH EXPR>)

                                        +  -> READ LINEAR     0009
                                        -  -> READ LINEAR     0010
            ----------------------------------------------

READ STATE   0008 IS INADEQUATE   THE FOLLOWING LOOK-AHEAD IS NECESSARY.

LOOK AHEAC ORD    0001  :        LA SYM NUM = 0001)

LA SET OF                    <ARITH EXPR>  -> REDUCE          0007 )
                                  DEFAULT  -> READ LINEAR     0008

STATE READ LINEAR        0008:    ACCESSING SYMBOL IS  <TERM>)

                                        *  -> READ LINEAR     0011
                                        /  -> READ LINEAR     0012
            ----------------------------------------------

STATE READ LINEAR        0009:    ACCESSING SYMBOL IS  +)

                         <NUMBER>  -> REDUCE                  0014
                     <IDENTIFIER>  -> REDUCE                  0015
```

```
                    -----------------------------------------------
                               <VARIABLE>  -> REDUCE                0013
                                   <TERM>  -> LOOK AHEAD ORD        0002
                                <PRIMARY>  -> REDUCE                0010

STATE READ LINEAR       0010:    ACCESSING SYMBOL IS  -)

                                 <NUMBER>  -> REDUCE                0014
                             <IDENTIFIER>  -> REDUCE                0015
                    -----------------------------------------------
                               <VARIABLE>  -> REDUCE                0013
                                   <TERM>  -> LOOK AHEAD ORD        0003
                                <PRIMARY>  -> REDUCE                0010

STATE READ LINEAR       0011:    ACCESSING SYMBOL IS  *)

                                 <NUMBER>  -> REDUCE                0014
                             <IDENTIFIER>  -> REDUCE                0015
                    -----------------------------------------------
                               <VARIABLE>  -> REDUCE                0013
                                <PRIMARY>  -> REDUCE                0011

STATE READ LINEAR       0012:    ACCESSING SYMBOL IS  /)

                                 <NUMBER>  -> REDUCE                0014
                             <IDENTIFIER>  -> REDUCE                0015
                    -----------------------------------------------
                               <VARIABLE>  -> REDUCE                0013
                                <PRIMARY>  -> REDUCE                0012

READ STATE   0013 IS INADEQUATE   THE FOLLOWING LOOK-AHEAD IS NECESSARY.

LOOK AHEAD ORD    0002  :       LA SYM NUM = 0001)

LA SET OF                    <ARITH EXPR>  -> REDUCE                0008
                                  DEFAULT  -> READ LINEAR           0013

STATE READ LINEAR       0013:    ACCESSING SYMBOL IS  <TERM>)

                                        *  -> READ LINEAR           0011
                                        /  -> READ LINEAR           0012
                    -----------------------------------------------
```

```
READ STATE   0014 IS INADEQUATE   THE FOLLOWING LOOK-AHEAD IS NECESSARY.

LOOK AHEAD ORD    0003  :       LA SYM NUM = 0001)
LA SET OF                <ARITH EXPR>  -> REDUCE           0009
                              DEFAULT  -> READ LINEAR      0014

STATE READ LINEAR       0014:    ACCESSING SYMBOL IS  <TERM>)
                                    *  -> READ LINEAR      0011
                                    /  -> READ LINEAR      0012
            ------------------------------------------------
```

```
THE GRAMMAR IS NOT LR(O).  LOOK-AHEAD MUST BE ADDED.

START COMPUTING LOOK-AHEAD SETS.
LOOK-AHEAD SETS HAVE BEEN COMPUTED.
THE GRAMMAR IS LR(1).

           T H E   L O O K - A H E A D   S E T S

NONTERMINAL     TERMINAL SYMBOLS
   SYMBOL      1   3   5   7   9
                 2   4   6   8   10
             ----------------------
   <PROGRAM> | 1
<STATEMENT L |   1                1
 <STATEMENT> |   1                1
<ASSIGNMENT> |     1
  <VARIABLE> |     1 1 1 1 1 1
      <EXPR> |     1
<ARITH EXPR> |     1   1 1
      <TERM> |     1   1 1 1 1
   <PRIMARY> |     1   1 1 1 1
```

```
START COMPUTING THE DPDA.
THE DPDA HAS BEEN COMPUTED,

THE DPDA FOR THE GRAMMAR CONSISTS OF THE TERMINAL TRANSITIONS OF THE CFSM PLUS
THE FOLLOWING REDUCE AND LOOK-BACK TRANSITIONS

STATE REDUCE 1:     POP 01          ->  READ LINEAR      0002
STATE REDUCE 2:     POP 00          ->  READ LINEAR      0003
STATE REDUCE 3:     POP 01          ->  READ LINEAR      0003
STATE REDUCE 4:     POP 01          ->  LOOK BACK        0000
STATE REDUCE 5:     POP 02          ->  READ LINEAR      0004
STATE REDUCE 6:     POP 00          ->  REDUCE           0005
STATE REDUCE 7:     POP 00          ->  LOOK AHEAD ORD   0000
STATE REDUCE 8:     POP 02          ->  LOOK AHEAD ORD   0000
STATE REDUCE 9:     POP 02          ->  LOOK AHEAD ORD   0000
STATE REDUCE 10:    POP 00          ->  LOOK BACK        0002
STATE REDUCE 11:    POP 02          ->  LOOK BACK        0002
STATE REDUCE 12:    POP 02          ->  LOOK BACK        0002
STATE REDUCE 13:    POP 00          ->  LOOK BACK        0003
STATE REDUCE 14:    POP 00          ->  LOOK BACK        0003
STATE REDUCE 15:    POP 00          ->  LOOK BACK        0001
STATE REDUCE 16:    POP 02          ->  EXIT             0000
STATE LOOK BACK 00:
                READ LINEAR     0001->  REDUCE           0002
                DEFAULT             ->  REDUCE           0003

STATE LOOK BACK 01:
                READ LINEAR     0001->  READ LINEAR      0005
                READ LINEAR     0003->  READ LINEAR      0005
                DEFAULT             ->  REDUCE           0013
```

```
STATE LCCK BACK 02:
                READ LINEAR         0006->  LOOK AHEAD ORD     0001
                READ LINEAR         0009->  LOOK AHEAD ORD     0002
                DEFAULT                 ->  LOOK AHEAD ORD     0003

STATE LOOK BACK 03:
                READ LINEAR         0011->  REDUCE             0011
                READ LINEAR         0012->  REDUCE             0012
                DEFAULT                 ->  REDUCE             0010
```

```
INTEGER NUMTERMINALS = 0010;
INTEGER NUMNTS = 0009;
INTEGER NUMSYMS = 0020;

GLOBAL DATA SEGTABLE BASE R12;

ARRAY 0133 BYTE VSTRING = (#C5X, #D9X, #D9X, #D6X, #D9X, #E2X, #E8X,
        #D4X, #40X, #6EX, #6DX, #4FX, #6DX, #4CX, #D7X, #D9X, #D6X,
        #C7X, #D9X, #C1X, #D4X, #6EX, #4CX, #E2X, #E3X, #C1X, #E3X,
        #C5X, #D4X, #C5X, #D5X, #E3X, #40X, #D3X, #C9X, #E2X, #E3X,
        #6EX, #C5X, #D5X, #C4X, #4CX, #E2X, #E3X, #C1X, #E3X, #C5X,
        #D4X, #C5X, #D5X, #E3X, #6EX, #4CX, #C1X, #E2X, #E2X, #C9X,
        #C7X, #D5X, #D4X, #C5X, #D5X, #E3X, #6EX, #5EX, #4CX, #E5X,
        #C1X, #D9X, #C9X, #C1X, #C2X, #D3X, #C5X, #6EX, #7EX, #4CX,
        #C5X, #E7X, #D7X, #D9X, #6EX, #4CX, #C1X, #D9X, #C9X, #E3X,
        #C8X, #40X, #C5X, #E7X, #D7X, #D9X, #6EX, #4CX, #E3X, #C5X,
        #D9X, #D4X, #6EX, #4EX, #60X, #4CX, #D7X, #D9X, #C9X, #D4X,
        #C1X, #D9X, #E8X, #6EX, #5CX, #61X, #4CX, #D5X, #E4X, #D4X,
        #C2X, #C5X, #D9X, #6EX, #4CX, #C9X, #C4X, #C5X, #D5X, #E3X,
        #C9X, #C6X, #C9X, #C5X, #D9X, #6EX);

ARRAY 0020 INTEGER LOCLENGTH = (#0000000A, #00000283, #00000983,
        #00001001, #000012C1, #00001901, #00001941, #00001BC1,
        #00001C01, #00001C48, #00001E4C, #00000349, #00000590,
        #00000A4B, #00000D0C, #0000104A, #00001306, #0000148C,
        #00001786, #00001989);

COMMENT  THE DPDA HAS 0015 READ STATES;

ARRAY 0015 SHORT INTEGER READSTART = (#0000S, #0002S, #0004S, #0006S,
        #0009S, #000BS, #000DS, #0010S, #0013S, #0016S, #0019S,
        #001CS, #001FS, #0022S, #0025S);

ARRAY 0015 BYTE RDNUM = (#01X, #01X, #01X, #02X, #01X, #01X, #02X,
        #02X, #02X, #02X, #02X, #02X, #02X, #02X, #02X);

ARRAY 0040 BYTE SYMLIST = (#01X, #00X, #0AX, #00X, #01X, #00X, #02X,
        #0AX, #00X, #03X, #00X, #04X, #00X, #09X, #0AX, #00X, #05X,
        #06X, #00X, #07X, #08X, #00X, #09X, #0AX, #00X, #09X, #0AX,
        #00X, #09X, #0AX, #00X, #09X, #0AX, #00X, #07X, #08X, #00X,
        #07X, #08X, #00X);

ARRAY 0040 SHORT INTEGER STATELIST = (#0001S, #06FFS, #020FS, #06FFS,
        #0210S, #06FFS, #0201S, #020FS, #06FFS, #0204S, #06FFS,
        #0006S, #06FFS, #020ES, #020FS, #06FFS, #0009S, #000AS,
        #06FFS, #000BS, #000CS, #06FFS, #020ES, #020FS, #06FFS,
        #020ES, #020FS, #06FFS, #020ES, #020FS, #06FFS, #020ES,
```

```
        #020FS, #06FFS, #000BS, #000CS, #06FFS, #000BS, #000CS,
        #06FFS);

COMMENT  THE DPDA HAS 0016 REDUCE STATES;

ARRAY 0017 BYTE NUMTOPOP = (#00X, #01X, #00X, #01X, #01X, #02X, #00X,
        #00X, #02X, #02X, #00X, #02X, #02X, #00X, #00X, #00X, #02X);

ARRAY 0017 SHORT INTEGER REDUCESUCC = (#5830S, #0002S, #0003S,
        #0003S, #0500S, #0004S, #0205S, #0300S, #0300S, #0300S,
        #0502S, #0502S, #0502S, #0503S, #0503S, #0501S, #0700S);

COMMENT  THE DPDA HAS 0004 LOOK AHEAD STATES;

ARRAY 0004 BYTE LASYMNUM = (#00X, #00X, #00X, #00X);

ARRAY 0004 SHORT INTEGER SUCCSTATE = (#0206S, #0207S, #0208S, #0209S);

ARRAY 0004 SHORT INTEGER FAILSTATE = (#0007S, #0008S, #000DS, #000ES);

ARRAY 0008 BYTE LATABLE = (
        COMMENT  <EXPR>;
        #10X, #00X,

        CCMMENT  <ARITH EXPR>;
        #16X, #00X,

        COMMENT  <ARITH EXPR>;
        #16X, #00X,

        COMMENT  <ARITH EXPR>;
        #16X, #00X);


COMMENT  THE DPDA HAS 0004 LOOKBACK STATES;

ARRAY 0004 BYTE LBSTART = (#00X, #02X, #05X, #08X);

ARRAY 0004 BYTE LBNUM = (#01X, #02X, #02X, #02X);

ARRAY 0011 SHORT INTEGER LBSTATE = (#0001S, #0000S, #0001S, #0003S,
        #0000S, #0006S, #0009S, #0000S, #000BS, #000CS, #0000S);

ARRAY 0011 SHORT INTEGER RESUMESTATE = (#0202S, #0203S, #0005S,
        #0005S, #020DS, #0301S, #0302S, #0303S, #020BS, #020CS,
        #020AS);
```

```
COMMENT  THE SYMBOLS ACCESSING THE STATES;

ARRAY 0015 BYTE SYMBEFOREREAD = (#00X, #01X, #0BX, #0CX, #0EX, #0FX,
         #04X, #11X, #12X, #05X, #06X, #07X, #08X, #12X, #12X);

ARRAY 0004 BYTE SYMBEFORELA = (#11X, #12X, #12X, #12X);

CLOSE BASE;

ARRAY 18 INTEGER SEGTABLE = (
         #00000000,
         #00000088,
         #000000D8,
         #000000F6,
         #00000105,
         #0000012E,
         #0000017E,
         #C0000190,
         #000001B2,
         #000001B6,
         #000001BE,
         #000001C6,
         #000001CE,
         #000001D2,
         #000001D6,
         #000001EC,
         #00000202,
         #00000211);
CCMMENT  END OF CARDS PUNCHED BY THE SLR(1) SYNTAX ANALYZER;

ALL DCNE.
```

APPENDIX D

PROTO-COMPILER LISTING

```
$TITLE   PROTOCOM
 BEGIN
 COMMENT  THIS PROGRAM IS A PROTO-COMPILER WRITTEN IN PL360
                                                                       ;
  COMMENT  ASSUME THE FOLLOWING CARDS WERE PUNCHED BY THE
           SLR(1) SYNTAX ANALYZER (SLR1ANAL);
  INTEGER NUMTERMINALS = 0010;
  INTEGER NUMNTS = 0009;
  INTEGER NUMSYMS = 0020;


  GLOBAL DATA SEGTABLE BASE R12;

  ARRAY 0133 BYTE VSTRING = (#C5X, #D9X, #D9X, #D6X, #D9X, #E2X, #E8X,
           #D4X, #40X, #6EX, #6DX, #4FX, #6DX, #4CX, #D7X, #D9X, #D6X,
           #C7X, #D9X, #C1X, #D4X, #6EX, #4CX, #E2X, #E3X, #C1X, #E3X,
           #C5X, #D4X, #C5X, #D5X, #E3X, #40X, #D3X, #C9X, #E2X, #E3X,
           #6EX, #C5X, #D5X, #C4X, #4CX, #E2X, #E3X, #C1X, #E3X, #C5X,
           #D4X, #C5X, #D5X, #E3X, #6EX, #4CX, #C1X, #E2X, #E2X, #C9X,
           #C7X, #D5X, #D4X, #C5X, #D5X, #E3X, #6EX, #5EX, #4CX, #E5X,
           #C1X, #D9X, #C9X, #C1X, #C2X, #D3X, #C5X, #6EX, #7EX, #4CX,
           #C5X, #E7X, #D7X, #D9X, #6EX, #4CX, #C1X, #D9X, #C9X, #E3X,
           #C8X, #40X, #C5X, #E7X, #D7X, #D9X, #6EX, #4CX, #E3X, #C5X,
           #D9X, #D4X, #6EX, #4EX, #60X, #4CX, #D7X, #D9X, #C9X, #D4X,
           #C1X, #D9X, #E8X, #6EX, #5CX, #61X, #4CX, #D5X, #E4X, #D4X,
           #C2X, #C5X, #D9X, #6EX, #4CX, #C9X, #C4X, #C5X, #D5X, #E3X,
           #C9X, #C6X, #C9X, #C5X, #D9X, #6EX);

  ARRAY 0020 INTEGER LOCLENGTH = (#0000000A, #00000283, #00000983,
           #00001001, #000012C1, #00001901, #00001941, #00001BC1,
           #00001C01, #00001C48, #00001E4C, #00000349, #00000590,
           #00000A4B, #00000D0C, #0000104A, #00001306, #0000148C,
           #00001786, #00001989);

  COMMENT  THE DPDA HAS 0015 READ STATES;

  ARRAY 0015 SHORT INTEGER READSTART = (#0000S, #0002S, #0004S, #0006S,
           #0009S, #000BS, #000DS, #0010S, #0013S, #0016S, #0019S,
           #001CS, #001FS, #0022S, #0025S);

  ARRAY 0015 BYTE RDNUM = (#01X, #01X, #01X, #02X, #01X, #01X, #02X,
```

```
        #02X, #02X, #02X, #02X, #02X, #02X, #02X, #02X);

ARRAY 0040 BYTE SYMLIST = (#01X, #00X, #0AX, #00X, #01X, #00X, #02X,
        #0AX, #00X, #03X, #00X, #04X, #00X, #09X, #0AX, #00X, #05X,
        #06X, #00X, #07X, #08X, #00X, #09X, #0AX, #00X, #09X, #0AX,
        #00X, #09X, #0AX, #00X, #09X, #0AX, #00X, #07X, #08X, #00X,
        #07X, #08X, #00X);

ARRAY 0040 SHORT INTEGER STATELIST = (#0001S, #06FFS, #020FS, #06FFS,
        #0210S, #06FFS, #0201S, #020FS, #06FFS, #0204S, #06FFS,
        #0006S, #06FFS, #020ES, #020FS, #06FFS, #0009S, #000AS,
        #06FFS, #000BS, #000CS, #06FFS, #020ES, #020FS, #06FFS,
        #020ES, #020FS, #06FFS, #020ES, #020FS, #06FFS, #020ES,
        #020FS, #06FFS, #000BS, #000CS, #06FFS, #000BS, #000CS,
        #06FFS);

COMMENT  THE DPDA HAS 0016 REDUCE STATES;

ARRAY 0017 BYTE NUMTOPOP = (#00X, #01X, #00X, #01X, #01X, #02X, #00X,
        #00X, #02X, #02X, #00X, #02X, #02X, #00X, #00X, #00X, #02X);

ARRAY 0017 SHORT INTEGER REDUCESUCC = (#0000S, #0002S, #0003S,
        #0003S, #0500S, #0004S, #0205S, #0300S, #0300S, #0300S,
        #0502S, #0502S, #0502S, #0503S, #0503S, #0501S, #0700S);

COMMENT  THE DPDA HAS 0004 LOOK AHEAD STATES;

ARRAY 0004 BYTE LASYMNUM = (#00X, #00X, #00X, #00X);

ARRAY 0004 SHORT INTEGER SUCCSTATE = (#0206S, #0207S, #0208S, #0209S);

ARRAY 0004 SHORT INTEGER FAILSTATE = (#0007S, #0008S, #000DS, #000ES);

ARRAY 0008 BYTE LATABLE = (
        COMMENT  <EXPR>;
        #10X, #00X,

        COMMENT  <ARITH EXPR>;
        #16X, #00X,

        COMMENT  <ARITH EXPR>;
        #16X, #00X,

        COMMENT  <ARITH EXPR>;
        #16X, #00X);
```

```
COMMENT  THE DPDA HAS 0004 LOOKBACK STATES;

ARRAY 0004 BYTE LBSTART = (#00X, #02X, #05X, #08X);

ARRAY 0004 BYTE LBNUM = (#01X, #02X, #02X, #02X);

ARRAY 0011 SHORT INTEGER LBSTATE = (#0001S, #0000S, #0001S, #0003S,
        #0000S, #0006S, #0009S, #0000S, #000BS, #000CS, #0000S);

ARRAY 0011 SHORT INTEGER RESUMESTATE = (#0202S, #0203S, #0005S,
        #0005S, #020DS, #0301S, #0302S, #0303S, #020BS, #020CS,
        #020AS);

COMMENT  THE SYMBOLS ACCESSING THE STATES;

ARRAY 0015 BYTE SYMBEFOREREAD = (#00X, #01X, #0BX, #0CX, #0EX, #0FX,
        #04X, #11X, #12X, #05X, #06X, #07X, #08X, #12X, #12X);

ARRAY 0004 BYTE SYMBEFORELA = (#11X, #12X, #12X, #12X);

CLOSE BASE;

ARRAY 18 INTEGER SEGTABLE = (
        #00000000,
        #00000088,
        #000000D8,
        #000000F6,
        #00000105,
        #0000012E,
        #0000017E,
        #00000190,
        #000001B2,
        #000001B6,
        #000001BE,
        #000001C6,
        #000001CE,
        #000001D2,
        #000001D6,
        #000001EC,
        #00000202,
        #00000211);
COMMENT  END OF CARDS PUNCHED BY THE SLR(1) SYNTAX ANALYZER;

INTEGER RESERVEDLIMIT = 0;
ARRAY 30 BYTE ALPHABET = ("ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#");

COMMENT  DECLARATIONS FOR THE SCANNER:
```

```
          TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE
          LAST SYMBOL SCANNED, CP IS THE POINTER TO THE LAST
          CHARACTER SCANNED IN THE CARD IMAGE, CBCD IS THE
          LAST SYMBOL SCANNED;

COMMENT   NUMBERVALUE CONTAINS THE NUMERIC VALUE OF THE LAST
          CONSTANT SCANNED;
INTEGER TOKEN = 1, PRODNUM, NUMBERVALUE, SP;
INTEGER REGISTER CP SYN R9;
ARRAY 64 BYTE BCD;
ARRAY 3 BYTE EOFIL = (#6DX, #4FX,#6DX);
ARRAY 12 BYTE IDENTS = ("<IDENTIFIER>");
ARRAY 8 BYTE NUMB = (#4CX, #D5X, #E4X, #D4X, #C2X, #C5X, #D9X, #6EX);
INTEGER DIVID = #00000061;
INTEGER LOCATION, LENGTH, VPT;
ARRAY 80 BYTE CBUF;  COMMENT  CARD BUFFER;
COMMENT  EXITFLAG IS USED TO INDICATE END OF COMPILING;
BYTE LISTFLAG, ENDIT, EXITFLAG = #00X;
COMMENT  XR IS THE ERROR ROUTINE PARAMETER REGISTER;
INTEGER REGISTER XR SYN R5;
SHORT INTEGER ERRCOUNT = 0S, CARDCOUNT = 0S;
INTEGER EOFILE = 0, NUMBER = 0,IDENT = 0, DIVIDE = 0;
LONG REAL CONWORK;  COMMENT  USED TO CONVERT TO DECIMAL;
BYTE TRUE = #FFX, FALSE = #00X;
SHORT INTEGER PREVIOUSERROR, ERRLIMIT = 50S;
ARRAY 132 BYTE BLANK = 132(" ");
ARRAY 132 BYTE WBUF;  COMMENT  WRITE BUFFER;
INTEGER MASK = #000000FF;
INTEGER MASK7 = #00000007;
INTEGER MASKFFFF = #0000FFFF;
INTEGER BLANKMASK = #40404040;
INTEGER MASK1 = #00000001;
INTEGER MASKFF00 = #0000FF00;
LONG REAL VR3;
INTEGER VR3HI SYN VR3(0);
INTEGER VR3LOW SYN VR3(4);
ARRAY 256 BYTE CHARTYPE = 256(1);
ARRAY 256 BYTE NOTLETTERORDIGIT = 256(1);
ARRAY 256 INTEGER TX;
INTEGER TEXTLIMIT = 71;  COMMENT  SCAN TO CBUF(TEXTLIMIT);
ARRAY 10 BYTE NUMS = ("0123456789");
BYTE PERIOD = #4BX;
ARRAY 8 BYTE CONBUF;
ARRAY 3 INTEGER TIME;
ARRAY 64 BYTE CBCD;
INTEGER SEGBASE, LATABSIZE, OFFSET;
```

```
FUNCTION SD(10,#FB00);

FUNCTION SETZONE(8,#96F0);  COMMENT  FUNCTION TO SET ZONE;
EXTERNAL PROCEDURE CLOSEM (R14);  NULL;

COMMENT  DECLARE USEFUL LITERALS TO SIMPLIGY
         ACCESSING THE PARSING TABLES;

   EQUATE AVSTRING        SYN 0;
   EQUATE ALOCLENGTH      SYN 4;
   EQUATE AREADSTART      SYN 8;
   EQUATE ARDNUM          SYN 12;
   EQUATE ASYMLIST        SYN 16;
   EQUATE ASTATELIST      SYN 20;
   EQUATE ANUMTOPOP       SYN 24;
   EQUATE AREDUCESUCC     SYN 28;
   EQUATE ALASYMNUM       SYN 32;
   EQUATE ASUCCSTATE      SYN 36;
   EQUATE AFAILSTATE      SYN 40;
   EQUATE ALATABLE        SYN 44;
   EQUATE ALBSTART        SYN 48;
   EQUATE ALBNUM          SYN 52;
   EQUATE ALBSTATE        SYN 56;
   EQUATE ARESUMSTATE     SYN 60;
   EQUATE ASYMBEFOREREAD SYN 64;
   EQUATE BSYMBEFORELA    SYN 68;


PROCEDURE FIND(R4);
   BEGIN ARRAY 4 INTEGER SAVEREGS;
      STM(R1,R4,SAVEREGS);
      MVC(63,BCD,BLANK);
      R6 := R6 - R6; R1 := VPT SHLL 2;
      R3 := ALOCLENGTH; R2 := SEGBASE + SEGTABLE(R3) + R1;
      R3 := B2; R1 := R3 SHRL 6;
      LOCATION := R1; R1 := R3 AND #3F;
      LENGTH := R1;  R2 := LENGTH - 1;
      FOR R1 := 0 STEP 1 UNTIL R2 DO
      BEGIN
         R3 := LOCATION + R1; IC(R6,VSTRING(R3));
         STC(R6,BCD(R1));
      END;
   LM(R1,R4,SAVEREGS);
   END;
```

```
PROCEDURE ERROR(R4); COMMENT  PRINTS AND ACCOUNTS FOR ALL
                              ERROR MESSAGES;
   BEGIN INTEGER SAVE4;
      SAVE4 := R4; RO := ERRCOUNT + 1; ERRCOUNT := RO;
      IF RO > ERRLIMIT THEN GOTO X;
      COMMENT  IF LISTING IS SUPPRESSED, FORCE PRINTING OF
               THE CARD BUFFER;
      IF ¬LISTFLAG THEN
      BEGIN
         RO := CARDCOUNT + 1; CVD(RO,CONWORK);
         UNPK(3,7,WBUF(45),CONWORK); SETZONE(WBUF(48));
         MVC(79,WBUF(52),CBUF); RO := @WBUF; WRITE;
         MVC(131,WBUF,BLANK);
      END;
      MVC(9,WBUF,"*** ERROR,");
      CASE XR OF
      BEGIN
         NULL;  COMMENT  CASE 1 NOT USED;
         BEGIN
            R1 := @WBUF(CP+27); MVI("|",B1);
            MVC(17,WBUF(11),"ILLEGAL CHARACTER:");
         END;
         BEGIN
            MVC(15,WBUF(11),"STACK OVERFLOW. ");
            SET(EXITFLAG);
         END;
         BEGIN
            R1 := @WBUF(CP+27); MVI("|",B1);
            MVC(19,WBUF(11),"ILLEGAL SYMBOL PAIR:");
         END;
         BEGIN
            MVC(24,WBUF(11),"PROGRAM ENDS PREMATURELY.");
            SET(EXITFLAG);
         END;
         BEGIN
            R1 := @WBUF(CP+27); MVI("|",B1);
            MVC(18,WBUF(11),"ILLEGAL IDENTIFIER:");
         END;
      END;
      RO := @WBUF; WRITE; MVC(131,WBUF,BLANK);
      IF EXITFLAG THEN GOTO EXIT;
      RO := ERRCOUNT;
      IF RO > 1 THEN
      BEGIN
         MVC(34,WBUF,"*** LAST ERROR DETECTED ON LINE  ");
         RO := PREVIOUSERROR; CVD(RO,CONWORK);
         UNPK(3,7,WBUF(33),CONWORK); SETZONE(WBUF(36));
         MVC(4,WBUF(37),". ***"); RO := @WBUF; WRITE;
```

```
      MVC(41,WBUF,BLANK);
    END;
    R0 := CARDCOUNT; PREVIOUSERROR := R0;
    R0 := ERRCOUNT;
    IF R0 = ERRLIMIT THEN
    BEGIN
      MVC(20,WBUF,"*** TOO MANY ERRORS, ");
      MVC(21,WBUF(20)," CHECKING ABORTED. ***");
    END;
X:  R4 := SAVE4;
  END;
PROCEDURE GETCARD(R4);
COMMENT  DOES ALL CARD READING AND LISTING;
  BEGIN INTEGER SAVE4;
    SAVE4 := R4; R0 := @CBUF; READ;
    IF ¬= THEN  COMMENT SIGNAL FOR EOF;
    BEGIN
      SET(ENDIT); MVC(79,CBUF,BLANK);
      MVC(10,CBUF,"EOF EOF EOF");
    END;
    COMMENT  CARDCOUNT PRINTED ON LISTING;
    R0 := CARDCOUNT + 1; CARDCOUNT := R0;
    IF LISTFLAG THEN
    BEGIN
      CVD(R0,CONWORK); UNPK(3,7,WBUF(20),CONWORK);
      SETZONE(WBUF(23)); MVC(79,WBUF(27),CBUF);
      R0 := @WBUF; WRITE; MVC(131,WBUF,BLANK);
    END;
    CP := 0; R4 := SAVE4;
  END;

PROCEDURE SCAN(R4);
  BEGIN INTEGER SAVE4, S1, S2;
    SAVE4 := R4; R0 := 0; NUMBERVALUE := R0;
    WHILE TRUE DO
    BEGIN
      IF CP > TEXTLIMIT THEN GETCARD ELSE
      BEGIN  COMMENT  BRANCH ON NEXT CHARACTER IN CBUF;
        IC(R1,CBUF(CP)); R1 := R1 AND MASK;
        IC(R2,CHARTYPE(R1)); R2 := R2 AND MASK;
        IF R2 < 10 THEN CASE R2 OF
        BEGIN
          COMMENT  CASE 1: ILLEGAL CHARACTER;
          BEGIN
            IF ENDIT THEN
            BEGIN
              R1 := R1; TOKEN := R1; GOTO L1;
            END;
```

```
  XR := 2; ERROR;
END;
COMMENT  CASE 2: BLANK;
BEGIN  COMMENT  SKIP OVER BLANKS;
  CP := CP + 1;
  IF CP <= TEXTLIMIT THEN
  BEGIN
    IC(R2,CBUF(CP)); R2 := R2 AND MASK;
    WHILE R2 = 64 AND CP < TEXTLIMIT DO
    BEGIN
      CP := CP +1; IC(R2,CBUF(CP));
    END;
  END;
  IF CP ¬= TEXTLIMIT THEN CP := CP -1;
END;
COMMENT  CASES 3 & 4 NOT USED;
NULL;
NULL;
COMMENT  CASE 5: A LETTER;
BEGIN
  S1 := CP; R1 := 0; S2 := R1;
  WHILE TRUE DO  COMMENT UNTIL END OF IDENT;
  BEGIN
    FOR CP := CP STEP 1 UNTIL TEXTLIMIT DO
    BEGIN
      IC(R1,CBUF(CP)); R1 := R1 AND MASK;
      IC(R2,NOTLETTERORDIGIT(R1));
      R2 := R2 AND MASK;
      IF R2 = 1 THEN
      BEGIN  COMMENT  END OF IDENTIFIER;
        IF CP > S1 THEN R2 := CP - S1
        ELSE R2 := TEXTLIMIT - S1 + CP + 1;
        R4 := 0;
        FOR R1 := S1 STEP 1 UNTIL CP DO
        BEGIN
          IC(R3,CBUF(R1)); STC(R3,CBCD(R4));
          R4 := R4 + 1;
        END;
      COMMENT  R2 IS LENGTH OF IDENT;
        R5 := NUMTERMINALS;
        IF R2 <= RESERVEDLIMIT THEN
          FOR R1 := 1 STEP 1 UNTIL R5 DO
          BEGIN
            VPT := R1; FIND; R3 := LENGTH - 1;
            IF R2 = LENGTH THEN
            BEGIN
              R6 := @CBCD - 1; R8 := @BCD -1;
              FOR R7 := 0 STEP 1 UNTIL R3 DO
```

```
                     BEGIN
                       R6 := R6 + 1; R8 := R8 + 1;
                       CLC(0,B6,B8);
                       IF ¬= THEN GOTO OUT;
                     END;
                     TOKEN := R1; GOTO L1;
                  OUT:
                  END;
               END;
         COMMENT  MUST BE <IDENT>;
            R1 := IDENT;
            IF R1 ¬= 0 THEN
            BEGIN
               TOKEN := R1; GOTO L1;
            END ELSE
            BEGIN
               XR := 6; ERROR; GOTO L2;
            END;
         END;
      END;
   COMMENT  END OF CARD;
      GETCARD;
   END;
END;
COMMENT  CASE 6: DIGIT;
BEGIN
   R1 := NUMBER; TOKEN := R1; R1 := R1 - R1;
   WHILE TRUE DO  COMMENT  UNTIL GOTO L1;
   BEGIN
      FOR CP := CP STEP 1 UNTIL TEXTLIMIT DO
      BEGIN
         IC(R1,CBUF(CP));
         IF R1 < 240 THEN GOTO L1;
         R3 := NUMBERVALUE * 10 + R1 - 240;
      END;
      GETCARD;
   END;
END;
COMMENT  CASE 7: /;
BEGIN
   R1 := DIVIDE; TOKEN := R1; CP := CP +1;
   GOTO L1;
END;
COMMENT  CASE 8: SPECIAL CHARACTER;
BEGIN
   R1 := R1 SHLL 2; R2 := TX(R1);
   TOKEN := R2; CP := CP +1; GOTO L1;
END;
```

```
          COMMENT  CASE 9: END OF FILE MARK,".";
          BEGIN
            R1 := 1; TOKEN := R1; GOTO L1;
          END;
        END;  COMMENT  END OF CASE ON CHARTYPE;
        L2:
        CP := CP +1;
      END;
  END;
L1:   R4 := SAVE4;
END;

PROCEDURE PRINTIME(R6);
  BEGIN INTEGER SAVE6;
    SAVE6 := R6;
    UNPK(7,3,CONBUF,TIME(0));
    MVC(1,WBUF(18),CONBUF(1));  MVC(0,WBUF(20),":");
    MVC(1,WBUF(21),CONBUF(3));
    R6 := SAVE6;
  END;
PROCEDURE PRINTDATE(R6);
  BEGIN INTEGER SAVE6, SAVE15;
    SAVE6 := R6; SAVE15 := R15;
    R1 := 2; SVC(11);
    R15 := SAVE15; R0 := R0 OR  #F;
    TIME(0) := R0;
    TIME(4) := R1; UNPK(7,3,CONBUF,TIME(4));
    MVC(1,WBUF(9),CONBUF(3));
    MVC(0,WBUF(11),PERIOD); MVC(2,WBUF(12),CONBUF(5));
    R6 := SAVE6;
  END;

PROCEDURE INITIALIZE(R4);
COMMENT  THIS PROCEDURE SETS VARIABLES TO BE USED IN THE
         PROGRAM.  IT SHOULD ALSO SAVE THE CURRENT TIME OF
         DAY TO BE USED IN PROCEDURE SUMMARIZE;
  BEGIN INTEGER SAVE4;
    SAVE4 := R4; MVC(131,WBUF,BLANK); R0 := @WBUF;
    MVC(34,WBUF,"   REPLACE THIS HEADING WITH ONE OF");
    MVC(36,WBUF(36),"YOUR OWN -- VERSION OF DECEMBER 1972.");
    WRITE; MVC(72,WBUF,BLANK); WRITE; WRITE; PRINTDATE;
    PRINTIME; MVC(8,WBUF,"TODAY IS ");
    MVC(0,WBUF(16),"@");
    R0 := @WBUF;
    WRITE; MVC(131,WBUF,BLANK); WRITE; WRITE;
    SEGBASE := R12;  COMMENT  SAVE BEGINNING ADDRESS OF
                              PARSING TABLES;
    R1 := NUMTERMINALS + 1 SHRL 3;
```

```
R2 := NUMTERMINALS + 1 AND #7;
IF R2 > 0 THEN R1 := R1 + 1;
LATABSIZE := R1;
COMMENT  SEARCH THE TERMINAL SYMBOLS FOR "_|_",
         "<NUMBER>", "<IDENT>", AND "/";

R5 := NUMTERMINALS;
FOR R1 := 1 STEP 1 UNTIL R5 DO
BEGIN
R4 := R4 - R4; R7 := R7 - R7; R3 := R3 - R3;
   VPT := R1; FIND; R2 := LENGTH;
   IF R2 > RESERVEDLIMIT THEN RESERVEDLIMIT := R2;
   IF R2 = 1 THEN
   BEGIN
      IC(R3,BCD(0));
      IF R3 < #C0 THEN
      BEGIN
         R4 := @CHARTYPE(R3); MVI(8,B4);
         R3 := R3 SHLL 2; TX(R3) := R1;
      END;
      IC(R3,BCD(0)); R3 := R3 AND MASK;
      IF R3 = DIVID THEN DIVIDE := R1;
   END ELSE
   IF R2 = 3 THEN
   BEGIN
      FOR R3 := 0 STEP 1 UNTIL 2 DO
      BEGIN
         IC(R4,BCD(R3)); IC(R7,EOFIL(R3));
         R4 := R4 AND MASK; R7 := R7 AND MASK;
         IF R4 ¬= R7 THEN GOTO YYY;
      END;
      EOFILE := R1;
   END ELSE
   IF R2 = 12 THEN
   BEGIN
      FOR R3 := 0 STEP 1 UNTIL 10 DO
      BEGIN
         IC(R4,BCD(R3)); IC(R7,IDENTS(R3));
         R4 := R4 AND MASK; R7 := R7 AND MASK;
         IF R4 ¬= R7 THEN GOTO YYY;
      END;
      IDENT := R1;
   END ELSE
   IF R2 = 8 THEN
   BEGIN
      FOR R3 := 0 STEP 1 UNTIL 7 DO
      BEGIN
         IC(R4,BCD(R3)); IC(R7,NUMB(R3));
```

```
               R4 := R4 AND MASK; R7 := R7 AND MASK;
               IF R4 ¬= R7 THEN GOTO YYY;
            END;
            NUMBER := R1;
         END;
      YYY:
      END;
      R5 := EOFILE;
      R5 := R5 SHLL 2; R1 := LOCLENGTH(R5) SHRL 6; R2 := @VSTRING(R1);
      MVC(2,B2,"EOF");
      R3 := R3 - R3; IC(R3," "); R4 := @CHARTYPE(R3);
      MVI(2,B4);

      R3 := R3 - R3;
      FOR R1 := 0 STEP 1 UNTIL 9 DO
      BEGIN
         IC(R3,NUMS(R1)); R4 := @CHARTYPE(R3); MVI(6,B4);
         R4 := @NOTLETTERORDIGIT(R3); MVI(0,B4);
      END;
      COMMENT  LENGTH OF ALPHABET IS 30;
      FOR R1 := 0 STEP 1 UNTIL 29 DO
      BEGIN
         R3 := R3 - R3; IC(R3,ALPHABET(R1));
         R4 := @CHARTYPE(R3); MVI(5,B4);
         R4 := @NOTLETTERORDIGIT(R3); MVI(0,B4);
         R3 := R3 SHLL 2; TX(R3) := R1;
      END;
      CP := TEXTLIMIT + 1; MVC(79,CBUF,BLANK); R1 := 0;
      SP := R1; SET(LISTFLAG); RESET(ENDIT); R4 := SAVE4;
   END;  COMMENT  END INITIALIZE;

PROCEDURE EMIT(R4); NULL;
COMMENT  THIS PROCEDURE HAS THE RESPONSIBILITY OF SETTING
         THE NEXT ELEMENT OF THE CODE ARRAY TO THE OPCODE
         DETERMINED BY PROCEDURE SYNTHESIZE;

PROCEDURE LOOKUP(R4); NULL;
COMMENT  THIS PROCEDURE LOOKS UP A NAME IN THE SYMBOL TABLE
         AND ENTERS IT IF NOT THERE;

PROCEDURE SYNTHESIZE(R4); NULL;
COMMENT THIS PROCEDURE IS RESPONSIBLE FOR THE SEMANTICS OF
         THE COMPILER.  IT TAKES THE FORM OF A LARGE CASE
         STATEMENT ON GLOBAL VARIABLE PRODNUM.  ARRIVE HERE
         FROM THE CASE STATEMENT IN PROCEDURE ANALYZE;

PROCEDURE PRINTSUMMARY(R4);
COMMENT  THIS PROCEDURE SHOULD SUBTRACT CURRENT TIME OF DAY
```

```
         WITH THAT SAVED IN PROCEDURE INITIALIZE AND STORE
         THE RESULT IN WBUF STARTING  IN COLUMN 19;
   BEGIN INTEGER SAVE4;
         INTEGER SAVE15;
         SAVE15 := R15;
         SAVE4  := R4;
      R1 := 2; SVC(11);
         R15 := SAVE15;
         RO := RO OR #F; TIME(8) := RO;
         SD(3,3,TIME(8),TIME(0));
         UNPK(7,3,CONBUF,TIME(8)); MVC(1,WBUF(21),CONBUF(3));
         MVC(0,WBUF(23),PERIOD); MVC(1,WBUF(24),CONBUF(5));
         MVC(6,WBUF(28),"SECONDS");
         MVC(17,WBUF,"TIME IN EXECUTION:"); RO := @WBUF; WRITE;
         MVC(40,WBUF,BLANK);
         R4 := SAVE4;
   END;

GLOBAL PROCEDURE ANALYZE(R4);
   BEGIN INTEGER SAVE4, NEXTSYMBOL;
         ARRAY 150 INTEGER STATESTACK = 150(0);
         INTEGER STATENUM =0, LASYMBOL = 0;

         PROCEDURE PUSHANDREAD(R4);
            BEGIN INTEGER SAVE4;
               SAVE4 := R4; R1 := SP;
               IF R1 < 150 THEN
               BEGIN
                  R1 := R1 + 1; SP := R1;
               END ELSE
               BEGIN
                  XR := 3; SET(EXITFLAG); ERROR;
               END;
               R1 := TOKEN; NEXTSYMBOL := R1;
               COMMENT  SET VAR(SP) TO BCD AND VAL(SP) TO
                         NUMBERVALUE;
               SCAN; R4 := SAVE4;
            END;  COMMENT  END PUSHANDREAD;

         INTEGER CYCLECNT = 0; SAVE4 := R4;
         WHILE TRUE DO
         BEGIN
            R1 := CYCLECNT + 1; CYCLECNT := R1;
            R1 := SP SHLL 2; R2 := STATESTACK(R1) SHRL 8 + 1;
            CASE R2 OF
            BEGIN
               COMMENT  CASE 1, READ VIA LINEAR SEARCH;
               BEGIN
```

```
   PUSHANDREAD; R1 := STATENUM SHLL 1;
   R2 := AREADSTART;
   R4 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B4);
   R1 := R1 SHRL 1; R5 := ARDNUM;
   R4 := SEGBASE + SEGTABLE(R5) + R1;
   IC(R3,B4); R3 := R3 AND MASK;
   R3 := R3 + R2; R4 := NEXTSYMBOL;
   R5 := ASYMLIST;
   R6 := SEGBASE + SEGTABLE(R5) + R2; IC(R5,B6);
   R5 := R5 AND MASK;
   WHILE R4 ¬= R5 AND R2 < R3 DO
   BEGIN
      R2 := R2 + 1; R6 := R6 + 1; IC(R5,B6);
   END;
   R2 := R2 SHLL 1;
   R4 := ASTATELIST;
   R3 := SEGBASE + SEGTABLE(R4) + R2;
   LH(R1,B3); R1 := R1 AND MASKFFFF;
   R2 := SP SHLL 2; STATESTACK(R2) := R1;
   R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 2, READ VIA AN ARRAY ACCESSS;
BEGIN
   PUSHANDREAD; R1 := STATENUM SHLL 1;
   R2 := AREADSTART;
   R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   R2 := R2 + NEXTSYMBOL SHLL 1; R1 := ASTATELIST;
   R3 := SEGBASE + SEGTABLE(R1) + R2; LH(R1,B3);
   R1 := R1 AND MASKFFFF;
   R2 := SP SHLL 2; STATESTACK(R2) := R1;
   R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 3, REDUCE;
BEGIN
   R1 := STATENUM; PRODNUM := R1; SYNTHESIZE;
   R1 := STATENUM; R2 := ANUMTOPOP;
   R3 := SEGBASE + SEGTABLE(R2) + R1;
   IC(R2,B3);
   R2 := R2 AND MASK; R3 := SP - R2; SP := R3;
   R1 := R1 SHLL 1; R2 := AREDUCESUCC;
   R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
   R1 := SP SHLL 2; STATESTACK(R1) := R2;
   R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 4, LOOK AHEAD (ORDINARY);
BEGIN
   R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
   R5 := STATENUM * LATABSIZE + R2;
```

```
  R3 := R1 AND MASK7; R4 := 7 - R3;
  R3 := ALATABLE;
  R6 := SEGBASE + SEGTABLE(R3) + R5; IC(R1,B6);
  R1 := R1 AND MASK SHRL R4 AND MASK1;
  IF R1 = 1 THEN
  BEGIN
    R1 := STATENUM SHLL 1;
    R2 := ASUCCSTATE;
    R3 := SEGBASE + SEGTABLE(R2) + R1;
    LH(R2,B3);
  END ELSE
  BEGIN
    R1 := STATENUM SHLL 1;
    R2 := AFAILSTATE;
    R3 := SEGBASE + SEGTABLE(R2) + R1;
    LH(R2,B3);
  END;
  R2 := R2 AND MASKFFFF; R1 := SP SHLL 2;
  STATESTACK(R1) := R2; R2 := R2 AND MASK;
  STATENUM := R2;
END;
COMMENT  CASE 5, LOOK AHEAD (FOR A PRODECTION
         WITH AN EMPTY RIGHT PART);
BEGIN
  R1 := TOKEN; LASYMBOL := R1; R2 := R1 SHRL 3;
  R5 := STATENUM * LATABSIZE + R2;
  R3 := R1 AND MASK7; R4 := 7 - R3;
  R3 := ALATABLE;
  R4 := SEGBASE + SEGTABLE(R3) + R5; IC(R1,B4);
  R1 := R1 AND MASK SHRL R4 AND MASK1;
  IF R1 = 1 THEN
  BEGIN
    R1 := SP SHLL 2;
    R2 := STATESTACK(R1) SHRL 8;
    WHILE R2 = 3 OR R2 = 4 DO
    BEGIN
      R3 := STATESTACK(R1) AND MASK SHLL 1;
      R4 := AFAILSTATE;
      R5 := SEGBASE + SEGTABLE(R4) + R3;
      LH(R4,B5);
      STATESTACK(R1) := R4;
      R2 := R4 SHRL 8;
    END;
    R1 := SP + 1; SP := R1;
    R1 := STATENUM SHLL 1;
    R2 := ASUCCSTATE;
    R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
  END ELSE
```

```
  BEGIN
    R1 := STATENUM SHLL 1;
    R2 := AFAILSTATE;
    R3 := SEGBASE + SEGTABLE(R2) + R1; LH(R2,B3);
  END;
  R1 := SP SHLL 2; STATESTACK(R1) := R2;
  R2 := R2 AND MASK; STATENUM := R2;
END;
COMMENT  CASE 6, LOOK BACK;
BEGIN
  R1 := SP - 1 SHLL 2; R2 := STATENUM;
  R3 := ALBSTART;
  R4 := SEGBASE + SEGTABLE(R3) + R2; IC(R3,B4);
  R3 := R3 AND MASK;
  R4 := ALBNUM;
  R5 := SEGBASE + SEGTABLE(R4) + R2; IC(R4,B5);
  R4 := R4 AND MASK + R3;
  R3 := R3 SHLL 1; R5 := ALBSTATE;
  R7 := SEGBASE + SEGTABLE(R5) + R3;
  LH(R5,B7);
  R6 := STATESTACK(R1); R3 := R3 SHRL 1;
  WHILE R6 ¬= R5 AND R3 < R4 DO
  BEGIN
    R3 := R3 + 1 SHLL 1; R5 := ALBSTATE;
  R7 := SEGBASE + SEGTABLE(R5) + R3; LH(R5,B7);
    R3 := R3 SHRL 1;
  END;
  R3 := R3 SHLL 1; R1 := ARESUMSTATE;
  R7 := SEGBASE + SEGTABLE(R1) + R3; LH(R1,B7);
  R2 := SP SHLL 2; STATESTACK(R2) := R1;
  R1 := R1 AND MASK; STATENUM := R1;
END;
COMMENT  CASE 7, ERROR;
BEGIN INTEGER PREVERRCYCLE = #FFFFFFFF;
  R0 := @WBUF; WRITE;
  R1 := CYCLECNT - 2; CYCLECNT := R1;
  IF R1 ¬= PREVERRCYCLE THEN
  BEGIN
    PREVERRCYCLE := R1; R1 := SP - 1 SHLL 2;
    R2 := STATESTACK(R1);
    IF R2 < 512 THEN
    BEGIN
      R2 := R2 AND MASK;
      R3 := ASYMBEFOREREAD;
      R4 := SEGBASE + SEGTABLE(R3) + R2;
      IC(R3,B4);
    END ELSE
    BEGIN
```

```
      R2 := STATENUM; R3 := BSYMBEFORELA;
      R4 := SEGBASE + SEGTABLE(R3) + R2; IC(R3,B4);
   END;
   R3 := R3 AND MASK; R2 := 33;
   FOR R7 := 0 STEP 1 UNTIL 1 DO
   BEGIN
     VPT := R3; FIND; R1 := LENGTH - 1;
     FOR R6 := 0 STEP 1 UNTIL R1 DO
     BEGIN
       IC(R5,BCD(R6)); STC(R5,WBUF(R2));
       R2 := R2 + 1;
     END;
     R2 := R2 + 2; R3 := STATENUM;
     IF R3 = 255 THEN R3 := NEXTSYMBOL
     ELSE R3 := LASYMBOL;
     R3 := R3 AND MASK;
   END;
   XR := 4; ERROR;
   MVC(131,WBUF,BLANK);
   MVC(17,WBUF,"PARTIAL PARSE IS: ");
   R0 := @WBUF; WRITE; MVC(17,WBUF,BLANK);
   R2 := SP - 1 SHLL 2;
   FOR R1 := 8 STEP 4 UNTIL R2 DO
   BEGIN
      R3 := STATESTACK(R1);
      IF R3 < 512 THEN
      BEGIN
         R3 := R3 AND MASK;
         R4 := ASYMBEFOREREAD;
         R5 := SEGBASE+SEGTABLE(R4)+R3;
         IC(R4,B5);
      END ELSE
      BEGIN
         R3 := R3 AND MASK;
         R4 := BSYMBEFORELA;
         R5 := SEGBASE+SEGTABLE(R4)+R3;
         IC(R4,B5);
      END;
      R4 := R4 AND MASK; VPT := R4; FIND;
      MVC(63,WBUF(4),BCD); WRITE;
      MVC(63,WBUF(4),BLANK);
   END;
END;
R1 := NEXTSYMBOL;
IF R1 = 1 THEN
BEGIN
   XR := 5; ERROR;
END ELSE
```

```
                  BEGIN
                     VPT := R1; FIND; R1 := LENGTH - 1;
                     MVC(16,WBUF,"THE INPUT SYMBOL,");
                     MVC(63,WBUF(20),BCD);
                     MVC(16,WBUF(60)," WILL BE IGNORED:");
                     R0 := @WBUF;
                     WRITE; MVC(131,WBUF,BLANK);
                  END;
                  R1 := STATENUM;
                  IF R1 = 255 THEN
                  BEGIN
                  COMMENT  ERROR OCCURRED IN A READ STATE;
                     R1 := SP - 1; SP := R1; R1 := R1 SHLL 2;
                     R2 := STATESTACK(R1) AND MASK;
                     STATENUM := R2;
                  END ELSE
                  BEGIN
                  COMMENT  ERROR OCCURRED IN A LOOK-AHEAD STATE;
                     SCAN;  COMMENT  SKIP THE NEXT SYMBOL;
                     R1 := R1 SHLL 1;
                  R2 := ASUCCSTATE;
                  R4 := SEGBASE + SEGTABLE(R2) + R1;
                     LH(R2,B4); R2 := R2 AND MASKFF00;
                     LH(R3,B4); R3 := R3 AND MASK;
                     R4 := ANUMTOPOP; R5 := SEGBASE+SEGTABLE(R4)+R3;
                     IC(R4,B5); R4 := R4 AND MASK;
                     IF R2 = 512 AND R4 ¬= 255 THEN
                        R1 := STATENUM OR #00000300
                     ELSE R1 := STATENUM OR #00000400;
                     R2 := SP SHLL 2; STATESTACK(R2) := R1;
                  END;
               END;  COMMENT  END OF CASE 7;
               BEGIN  COMMENT EXIT;
                  R1 := 1; TOKEN := R1; R1 := 0;
                  SP := R1; STATESTACK(R1) := R1;
                  STATENUM := R1; GOTO XXX;
               END;
            END;  COMMENT  END OF CASE(STATETYPE);
         END;
XXX:     R4 := SAVE4;
      END;

PROCEDURE MAIN(R4);
   BEGIN INTEGER SAVE4;
      SAVE4 := R4;
      INITIALIZE;
      ANALYZE;
      PRINTSUMMARY;
```

```
      R4 := SAVE4;
    END;

MAIN;
EXIT:
MVC(17,WBUF,"END OF COMPILATION"); R0 := @WBUF; WRITE;
END.
```

APPENDIX E

PROTO-COMPILER OUTPUT

```
   REPLACE THIS HEADING WITH ONE OF YOUR OWN -- VERSION OF DECEMBER 1972.

TODAY IS 72.339 @ 17:34

                   0001        XRAY = 1; YELLOW = 1  XRAY*XRAY;
                                                        |

*** ERROR, ILLEGAL SYMBOL PAIR:  <TERM>  <IDENTIFIER>
PARTIAL PARSE IS:
    <STATEMENT LIST>
    <VARIABLE>
    =
    <TERM>
THE INPUT SYMBOL,   <IDENTIFIER>                              WILL BE IGNORED:

                  0002        ZEBRA = 2/1 + YELLOW;    ABLE = ZEBRA *1+ZEBRA-
                  0003        BETA = ABLE / XRAY + ZEBRA / XRAY;  END
                  0004    EOF EOF EOF
TIME IN EXECUTION:  00.02  SECONDS
END OF COMPILATION
```

APPENDIX F

```
*************************************************************
*           OS/360 OPERATING SYSTEM INTERFACE FOR PL360
*************************************************************

         ICTL    1,71,18
         SPACE
*************************************************************
*************************************************************
         SPACE
         MACRO
&EP      ENTER
         ENTRY   &EP
         USING   &EP,15
&EP      STM     12,2,SAVE             SAVE REGISTERS
         L       12,=A($PL360IO)       ESTABLISH ADDRESSING
         USING   $PL360IO,12
         DROP    15
         MEND
         SPACE
         MACRO
         EXIT
         LM      12,2,SAVE             RESTORE REGISTERS
         BR      14
         DROP    12
         MEND
         SPACE   2
$PL360IO CSECT
         SPACE
LINELEN  EQU     132                   PRINTER LINE LENGTH
LINESMAX EQU     60                    PRINTER LINES/PAGE
         PRINT   NOGEN
         SPACE
*   GLOBAL PROCEDURE READ(R14)
         (R0)  = BUFFER ADDRESS
         (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
READ     ENTER
         TM      SYSIN+DOPEN,OPENMASK  TEST FOR OPEN DCB
         BO      READ1
         LR      2,0
         OPEN    (SYSIN,(INPUT))       ISSUE OPEN
         LR      0,2
READ1    CLI     EOF,X'FF'             TEST FOR PREVIOUS END
         BE      ERRPROC
         GET     SYSIN,(0)             GET CARD
READ2    CLI     EOF,0                 SET CONDITION CODE
         EXIT
         SPACE
*                  EOD EXIT ROUTINE
         USING   $PL360IO,12
ENDRDR   MVI     EOF,X'FF'             EODAD EXIT
         B       READ2
         DROP    12
         SPACE   2
*   GLOBAL PROCEDURE WRITE(R14)
*        (R0)  = BUFFER ADDRESS
*        (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
WRITE    ENTER
         LR      2,0
         TM      SYSOUT+DOPEN,OPENMASK TEST FOR OPEN DATA SE
         BO      WRITE1
         OPEN    (SYSOUT,(OUTPUT))
WRITE1   PUT     SYSOUT,(1)            GET NEXT BUFFER ADDRE
         MVC     0(1,1),CARRCONT       SUPPLY CONTROL CHARAC
         CLI     CARRCONT,C'1'
         BNE     WRITE2
         MVI     LINECNT,0             RESET LINE COUNT AND
         MVI     CARRCONT,C' '
```

```
WRITE2   MVC      1(LINELEN,1),0(2)     TRANSFER BUFFER
         IC       2,LINECNT             INCREMENT LINE COUNT
         LA       2,1(,2)
         STC      2,LINECNT
         CLI      LINECNT,LINESMAX      TEST FOR FULL PAGE
         BL       WRITE3
         MVI      CARRCONT,C'1'         SET SKIP
WRITE3   LM       12,2,SAVE
         BR       14
         DROP     12
         SPACE    2
*   GLOBAL PROCEDURE PAGE(R14)
*        (R14) = RETURN ADDRESS
         ENTRY    PAGE
         USING    PAGE,15
PAGE     MVI      CARRCONT,C'1'         SET
         BR       14
         SPACE    2
*   GLOBAL PROCEDURE PUNCH(R14)
*        (R0)  = BUFFER ADDRESS
*        (R13) = SAVE AREA ADDRESS
*        (R14) = RETURN ADDRESS
PUNCH    ENTER
         TM       SYSPUNCH+DOPEN,OPENMASK
         BO       PUNCH1
         LR       2,0
         OPEN     (SYSPUNCH,(OUTPUT))
         LR       0,2
PUNCH1   PUT      SYSPUNCH,(0)          PUT CARD IMAGE
         EXIT
         SPACE    2
SYSOUT   DCB      DSORG=PS,MACRF=PL,DDNAME=SYSPRINT,DEVD=DA,R
                  LRECL=LINELEN+1,BFTEK=S,EROPT=ABE
SYSIN    DCB      DSORG=PS,MACRF=GM,DDNAME=SYSIN,DEVD=DA,RECF
                  LRECL=80,BFTEK=S,EROPT=ABE,EODAD=ENDRDR
SYSPUNCH DCB      DSORG=PS,MACRF=PM,DDNAME=SYSPUNCH,DEVD=DA,R
                  LRECL=80,BFTEK=S,EROPT=ABE
SAVE     DS       7F
ERRPROC  DC       H'0'
EOF      DC       X'00'
LINECNT  DC       X'00'                     PRINTER LINE COUN
CARRCONT DC       C'1'                  PRINTER CARRIAGE CONT
         SPACE
         DCBD
$PL360IO CSECT
DOPEN    EQU      DCBOFLGS-IHADCB       BYTE SET BY OPEN
OPENMASK EQU      X'10'
         END
```

APPENDIX G

JOB CONTROL LANGUAGE

```
**************************************
* JCL TO EXECUTE THE SYNTAX ANALYZER *
**************************************

  //W000564    JOB (0564,0812NT,CS14),'WOODS'
  //JOBLIB     DD DSN=C0012,PL360,DISP=OLD,VOL=SER=MARY,UNIT=2314
  //COMP       EXEC PGM=PL360,REGION=150K
  //SYSPRINT   DD SYSOUT=A,SPACE=TRK,(10,5)),
  //           DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
  //SYSGO      DD UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE),DISP=(,PASS),
  //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
  //SYSIN      DD *

       COMMENT  THE SYNTAX ANALYZER;

  //LINK       EXEC PGM=IEWL,PARM='MAP,LIST',REGION=100K,
               COND=(0,NE,COMP)
  //SYSUT1     DD UNIT=SYSDA,SPACE=(TRK,(20,10))
  //SYSLMOD    DD DSN=&T(PROGRAM),UNIT=SYSDA,DISP=(,PASS),
  //           SPACE=(CYL,(1,1,1),RLSE)
  //SYSLIB     DD DSN=S0938.PLIO,UNIT=2314,VOL=SER=MARY,DISP=SHR
  //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
  //           SPACE=(TRK,5)
  //SYSLIN     DD DSN=*.COMP.SYSGO,DISP=(OLD,DELETE)
  //           DD *
  //GO         EXEC PGM=*.LINK.SYSLMOD,REGION=100K,COND=(4,LT,LINK)
  //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
  //           SPACE=(TRK,(10,5))
  //SYSPUNCH   DD SYSOUT=B,DCB=BLKSIZE=800
  //SYSIN      DD *
  DATA CARDS.
  /*
```

```
**************************************
* JCL TO EXECUTE THE PROTO-COMPILER  *
**************************************

  //W000564    JOB (0564,0812NT,CS14),'WOODS'
  //JOBLIB     DD DSN=C0012,PL360,DISP=OLD,VOL=SER=MARY,UNIT=2314
  //COMP       EXEC PGM=PL360,REGION=100K
  //SYSPRINT   DD SYSOUT=A,SPACE=TRK,(10,5)),
  //           DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
  //SYSGO      DD UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE),DISP=(,PASS),
  //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
  //SYSIN      DD *

          COMMENT  THE PROTO-COMPILER

  //LINK       EXEC PGM=IEWL,PARM='MAP,LIST',REGION=100K,
               COND=(0,NE,COMP)
  //SYSUT1     DD UNIT=SYSDA,SPACE=(TRK,(20,10))
  //SYSLMOD    DD DSN=&T(PROGRAM),UNIT=SYSDA,DISP=(,PASS),
  //           SPACE=(CYL,(1,1,1),RLSE)
  //SYSLIB     DD DSN=S0938.PLIO,UNIT=2314,VOL=SER=MARY,DISP=SHR
  //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
  //           SPACE=(TRK,5)
  //SYSLIN     DD DSN=*.COMP.SYSGO,DISP=(OLD,DELETE)
  //           DD *
  //GO         EXEC PGM=*.LINK.SYSLMOD,REGION=60K,COND=(4,LT,LINK)
  //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
  //           SPACE=(TRK,(10,5))
  //SYSIN      DD *
  DATA CARDS.
  /*
```

LIST OF REFERENCES

1. Wirth, N., "PL360, A Programming Language for the 360 Computers," J. ACM, v. 15, p. 37-74, 1968.

2. Blanchard, R. C., Steps Toward a PL360-Based Compiler Generator for the IBM 360 Computer, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1972.

3. DeRemer, F. L., "Simple LR(k) Grammars," Comm. ACM, v. 14, p. 453-460, 1971.

4. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator, Prentice-Hall, Englewood Cliffs, N. J., 1970.

5. Feldman, J., and Gries, D., "Translator Writing Systems," Comm. ACM, v. 11, p. 77-113, 1968.

6. Wirth, N., and Weber, H., "EULER - A generalization of ALGOL, and its Formal Definition," Comm. ACM, v. 9, p. 13-25, p. 89-99, 1966.

7. OS/360 PL360 Compiler, IBM Contributed Library (Type IV) Program, Number 360D-03.2.011.

8. Malcolm, M. A., PL360 (Revised), A Programming Language for the IBM 360, Stanford Univ., May 1971.

9. CEP Rep. 2, Simple LR(k) Grammars: Definition and Implementation, by R. L. DeRemer, U. of Calif., Santa Cruz, 4 September 1970.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Instructor R. H. Brubaker, Code 53Bh Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Asst. Professor G. E. Heidorn, Code 55Hd Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	Asst. Professor G. A. Kildall, Code 53Kd Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
6.	LT R. A. Woods, USN 212 North Park Drive Hutchinson, Kansas 67501	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3 REPORT TITLE

A PL360-Based Compiler Generating System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Master's Thesis; December 1972

5 AUTHOR(S) *(First name, middle initial, last name)*

Robert Allen Woods

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1972	112	9

8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

A compiler generating system written in the language PL360 to run on IBM System/360 computers is presented. The concepts and principles of the XPL compiler generating system are reviewed. The SLR(k) parsing algorithm is briefly described, and an example of SLR(1) parsing is presented. A description of the compiler generating system is presented along with its limitations, and instructions for its use are given. The required PL360 to System/360 interface is described and a listing is included. Program listings and sample input and output are included in the appendices.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Compiler Generator						
PL360						
Proto-Compiler						
Syntax Analyzer						
Syntax Checker						
SLR(k) Parser						

141857

Thesis
W8435 Woods
c.1 A PL360-based compiler
generating system.